

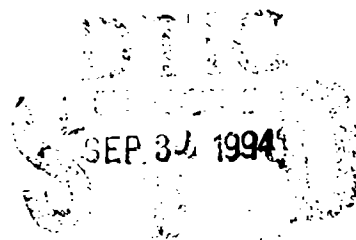
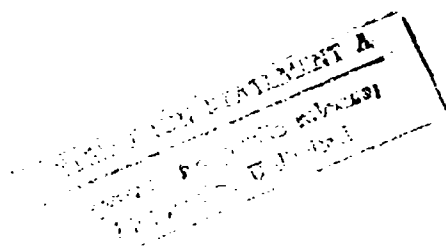
AD-A285 086



TASK: UU04
CDRL: 05203
30 November 1992

PCTE Browser Tool (PBT) Version 1.0

Informal Technical Data



STARS-UC-05203/001/00
30 November 1992

94-31235



STARS-UC-05203/001/00

94 0 0 0 0

TASK: UU04
CDRL: 05203
30 November 1992

USER MANUAL
For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

PCTE Browser Tool (PBT)
Version 1.0
SunOS implementation

STARS-UC-05203/001/00
30 November 1992

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0011

Prepared for:
Electronic Systems Center
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:
Paramax Systems Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

Accession No.	
DTIC	<input checked="" type="checkbox"/>
DTIC	<input type="checkbox"/>
Unclas	<input type="checkbox"/>
Just	<input type="checkbox"/>
By <i>perform 50</i>	
Date	
A-1	

Data ID: STARS-UC-05203/001/00

Distribution Statement "A"
per DoD Directive 5230.24

Authorized for public release; Distribution is unlimited.

Copyright 1992, Paramax Systems Corporation, Reston, Virginia
Copyright is assigned to the U.S. Government, upon delivery thereto, in accordance with
the DFAR Special Works Clause.

Developed by: Paramax Systems Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

Permission to use, copy, modify, and comment on this document for purposes stated under Distribution "A" and without fee is hereby granted, provided that this notice appears in each whole or partial copy. This document retains Contractor indemnification to The Government regarding copyrights pursuant to the above referenced STARS contract. The Government disclaims all responsibility against liability, including costs and expenses for violation of proprietary rights, or copyrights arising out of the creation or use of this document.

In addition, the Government, Paramax, and its subcontractors disclaim all warranties with regard to this document, including all implied warranties of merchantability and fitness, and in no event shall the Government, Paramax, or its subcontractor(s) be liable for any special, indirect or consequential damages or any damages whatsoever resulting from the loss of use, data, or profits, whether in action of contract, negligence or other tortious action, arising in connection with the use or performance of this document.

TASK: UU04
CDRL: 05203
30 November 1992

USER MANUAL
PCTE Browser Tool (PBT)
Version 1.0
SunOS implementation

Principal Author(s):

Michael J. Horton, Valley Forge Labs

Date

Approvals:

Task Manager *Dr. Paul Orgren*

Date

(Signatures on File)

TASK: UU04
CDRL: 05203
30 November 1992

USER MANUAL
PCTE Browser Tool (PBT)
Version 1.0
SunOS implementation

Change Record:

<i>Data ID</i>	<i>Description of Change</i>	<i>Date</i>	<i>Approval</i>
STARS-UC-05203/001/00	Successor volume: Upgrade for software version 1.0	30 November 1992	<i>on file</i>
STARS-TC-04014/003/00	Original Issue: PCTE Browser Tool (PBT) User Manual for PBT version 0.1	12 June 1992	<i>on file</i>

TASK: UU04
CDRI: 05203
30 November 1992

USER MANUAL
For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

PCTE Browser Tool (PBT)
Version 1.0
SunOS implementation

STARS-UC-05203/001/00
30 November 1992

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0011

Prepared for:
Electronic Systems Center
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:
Paramax Systems Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

Contents

1 INTRODUCTION	1
1.1 SCOPE	1
1.2 PBT Overview	1
2 References	5
3 INSTRUCTIONS FOR USE	6
3.1 PREPARATIONS FOR USE	6
3.1.1 Installing the PBT Executable	6
3.1.2 Installing the pbt SDS	6
3.1.3 Installing the X Resource Files	7
3.1.3.1 Identifying the X Resource Files	7
3.1.4 Starting the X Window System	8
3.1.5 Starting the PCTE Server	8
3.1.6 Logging into PCTE	8
3.1.7 Executing the PCTE Browser Tool	8
3.1.7.1 Order of Evaluation of Command Line Arguments	9
3.2 BROWSER LOOK AND FEEL	10
3.2.1 Initial Browser Window	10
3.2.2 Composite View Windows	12
3.2.3 Local View Window	14
3.2.4 Dialog Boxes	14
3.2.5 Control Panels	16
3.2.6 Text Windows	17
3.2.7 Alert Boxes	18
3.2.8 Confirmation Boxes	19
3.2.8.1 Use of View's Topology Region	19
3.2.9 Miscellaneous Window Manager Operations	19
4 PCTE Browser Capabilities	20
4.1 Object Identification	20
4.1.1 Aliases in Object Identification	21
4.1.1.1 Displaying Object Aliases	22
4.1.1.2 Setting Object Aliases	23
4.1.1.3 Deleting Object Aliases	23
4.1.1.4 Saving Object Aliases	23
4.1.1.5 Dynamic Loading of Object Aliases	24
4.1.1.6 Loading Object Aliases at PBT Session Startup	24
4.2 Name Pattern Syntax	25
4.3 Creation/Maintenance of View Windows	25
4.3.1 Creation of Composite View Windows	26
4.3.1.1 Dynamic Creation of Composite Views	26
4.3.1.2 Creating Composite Views at PBT Session Startup	27

4.3.2	Creation of Local View Windows	28
4.3.2.1	Dynamic Creation of Local Views	29
4.3.2.2	Creating Local Views at PBT Session Startup	30
4.3.3	Creation of Topology Regions in View Windows	31
4.4	Filtering Objects and Links Within View Windows	31
4.4.1	Hiding/Revealing Objects Based Upon Object Kind	31
4.4.2	Hiding/Revealing Links Based Upon Link Category	32
4.4.3	Hiding/Revealing Objects Based Upon Object Type Name	32
4.4.4	Hiding/Revealing Links Based Upon Link Type Name	34
4.5	A Priori Filtering of Links	34
4.5.1	Displaying Link Type Name Patterns	35
4.5.2	Adding Link Type Name Patterns	37
4.5.3	Deleting Link Type Name Patterns	37
4.5.4	Saving Link Type Name Patterns	37
4.5.5	Loading Link Type Name Patterns	38
4.5.6	Loading Link Type Name Patterns at PBT Session Startup	38
4.6	Current Object-Oriented Operations	39
4.6.1	Setting the Current Object	39
4.6.1.1	Dynamic Setting of the Current Object	39
4.6.1.2	Setting of the Current Object at PBT Session Startup	40
4.6.2	Displaying the Current Object	40
4.7	Current Working Schema-Oriented Operations	41
4.7.1	Setting the Current Working Schema	41
4.7.1.1	Dynamic Setting of the Current Working Schema	41
4.7.1.2	Setting the Current Working Schema at PBT Session Startup	42
4.7.2	Displaying the Current Working Schema	43
4.7.3	Displaying SDSes in Metabase	43
4.8	Object-Oriented Operations	43
4.8.1	Displaying the Pathname	45
4.8.2	Displaying Object Attributes	45
4.8.3	Displaying PCTE File Contents	45
4.8.4	Invoking Actions on PCTE Objects	47
4.9	Link-Oriented Operations	48
4.9.1	Displaying Non-Key Link Attributes	49
4.9.2	Displaying Full Linkname	49
4.9.3	Navigating To Source or Target Object	50
4.10	Displaying the Latest PCTE Exception Information	50
4.11	Customizing the Browser	50
4.12	Terminating the Browser Session	55
A	Appendix: Customizing the PBT X Resources	56
B	Appendix: FCTE Browser Tool Help Message	64

List of Figures

1	Graph of PCTE Objects and Links	2
2	Topology Region Within View	4
3	Initial Browser Window	10
4	Objects Command Button Menu	11
5	Composite View Window Example	13
6	Local View Window Example	15
7	Dialog Box Example	16
8	Control Panel Example	17
9	Scrollable Text Window Example	18
10	Alert Box Example	19
11	Confirmation Box Example	19
12	Object Aliases Submenu	22
13	Display Aliases Text Window	22
14	Set Alias Name Control Panel	23
15	Delete Alias Dialog Box	23
16	Save Aliases Dialog Box	24
17	Load Aliases Dialog Box	24
18	Display Command Button Menu	26
19	Composite View Control Panel	27
20	Local View Control Panel	29
21	Filter Command Button Menu	32
22	[Un]Suppress by Object Kind Submenu	32
23	[Un]Suppress by Link Category Submenu	33
24	Suppress Object Name Dialog Box	33
25	Unsuppress Object Name Dialog Box	33
26	Suppress Link Name Dialog Box	34
27	Unsuppress Link Name Dialog Box	34
28	Links Menu	36
29	Display Patterns Text Window	36
30	Add Pattern Dialog Box	37
31	Delete Pattern(s) Dialog Box	37
32	Save Patterns Dialog Box	38
33	Load Patterns Dialog Box	38
34	Current Object Menu	39
35	Set the Current Object Dialog Box	40
36	Current Object Pathname Text Window	40
37	SDSes Submenu	41
38	Set PCTE Working Schema Dialog Box	42
39	Current Working Schema Text Window	43
40	SDSes in Metabase Text Window	44
41	Non-File Object Menu	44
42	File Object Menu	45
43	Object Pathname Text Window	45
44	Object Attribute Values Text Window	46

45	PCTE File Contents Text Window	47
46	Viewer Selection Control Panel	47
47	Object Menu With Actions	48
48	Action Selection Control Panel	48
49	Link Operations Menu	49
50	Link Attribute Values Text Window	49
51	Full Linkname Text Window	50
52	Graph Including SDS Names	52
53	Setup Control Panel	53

1 INTRODUCTION

1.1 SCOPE

The PCTE Browser Tool (PBT) is an instance of the Paramax STARS Reusable Graphical Browser (RGB), a generic graphical browser for the display of networks of nodes and arcs. In the case of the PBT, the nodes displayed by the RGB are PCTE *objects*, and the arcs are PCTE *links*. The PBT is a Motif-oriented X Window System application making use of the SA-Motif Ada bindings, a commercial product of Systems Engineering Research Corporation (SERC).

This tool has been developed in Ada using the Sun Ada 1.0 compilation system from Sun Microsystems.

The PBT is ultimately intended for use in an ECMA PCTE environment and has been implemented using the ECMA-162 Ada programming bindings to PCTE. However, in the absence of a conforming ECMA PCTE implementation, the PBT has been built on top of the Emeraude V12 PCTE 1.5 implementation, using a subset implementation of the ECMA Ada binding developed by Paramax STARS (release 0.2).

This document describes the general capabilities of the PBT and how to invoke them. For detailed instructions on how to install or rebuild the PBT, consult the associated Version Description Document (VDD).

1.2 PBT Overview

The PBT is designed to graphically display selected parts of a PCTE object base. Graphs of objects in the object base and the relationships amongst them are created and displayed at the PBT user's request. The PBT is intended to complement text-oriented commands such as `obj_list_links` and `obj_list_attr`, which are included with the Emeraude V12 PCTE release—commands intended to be invoked from the text-oriented `esh` command shell.

The PBT allows the user to create and display any number of networks of objects—*Views* in PBT terminology—from a single PCTE object base. Figure 1 illustrates one such View.

In figure 1, two basic kinds of objects are represented, using two different icons: PCTE *file* objects use a representation of a sheet of paper filled with text as an icon, while ordinary PCTE objects without contents use a representation of a file folder. Most objects in the PCTE object base are specializations of one of the basic PCTE object types. The PBT identifies the specialization by labeling each object with the object type name as defined in the current *working schema*. It also labels each arc with the link name and key value associated with that PCTE link. The PBT distinguishes amongst the various kinds of links by decorating the arcs with different icons for different link categories. In figure 1, *composition* links are labeled with *C*'s, *reference* links with *R*'s, and *implicit* links with *I*'s. When run on color monitors, the PBT uses different colors for the different icons to further

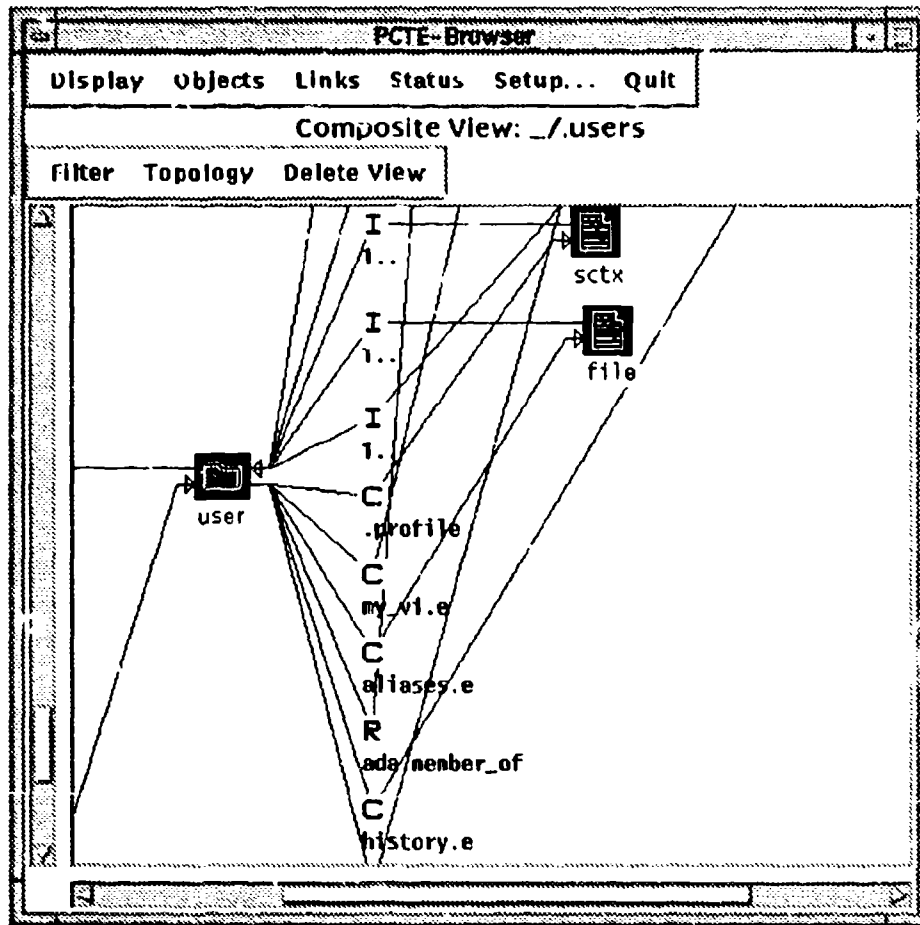


Figure 1: Graph of PCTE Objects and Links

distinguish amongst the basic node and arc kinds.

The PBT supports two types of Views:

- Composite Views
- Local Views

A *Composite View* includes the set of PCTE objects making up a composite object, i.e., the set of objects which comprise the tree formed by traversing the PCTE *composition* links emanating from a specified *root* object. The user can specify that the entire tree of composition links be included in the View. Alternatively—presumably for “large” composite objects—the user can specify that the tree be cut off at a user-specifiable maximum depth. Under the user’s control, a Composite View may also include other types of (internal and external) PCTE links emanating from the individual PCTE objects comprising the composite object.

A *Local View* consists of the set of PCTE objects which fall within a user-specified maximum distance from a specified *start* object. The distance from the start object is defined as the minimum number of link traversals beginning at the start object to reach another object in the set. Each Local View also includes all the links amongst the objects in the set.

The objects to be used as the roots of Composite Views or the start objects for Local Views can be identified by the user via their PCTE pathnames. In addition, once Local and Composite Views exist, these objects can also be selected from amongst the objects within these existing Views via point-and-click mouse-oriented operations.

The PBT supports a number of operations on objects and links displayed in Views:

For every kind of PCTE object, it is possible to:

- display the object's pathname within that View;
- display its attributes;
- make it the PBT session's *Current Object*;
- create a new Composite View rooted at the object;
- create a new Local View starting from the object;
- invoke one of the associated *action* programs, if any exist for the object.

In addition, for every PCTE *file* object, it is possible to:

- display its contents.

For every category of PCTE link, it is possible to:

- display the full pathname within that View of the link's target object;
- display any non-key link attributes;
- "go to" the source of the link by scrolling the graph so that the source object is approximately centered within the View window;
- "go to" the target of the link by scrolling the graph so that the target object is approximately centered within the View window.

In addition, for every specialized *pbt-viewer* and *pbt-action* link, it is possible to:

- invoke the program that is the link target, passing it the pathname of link source so that it can have access to the source object.

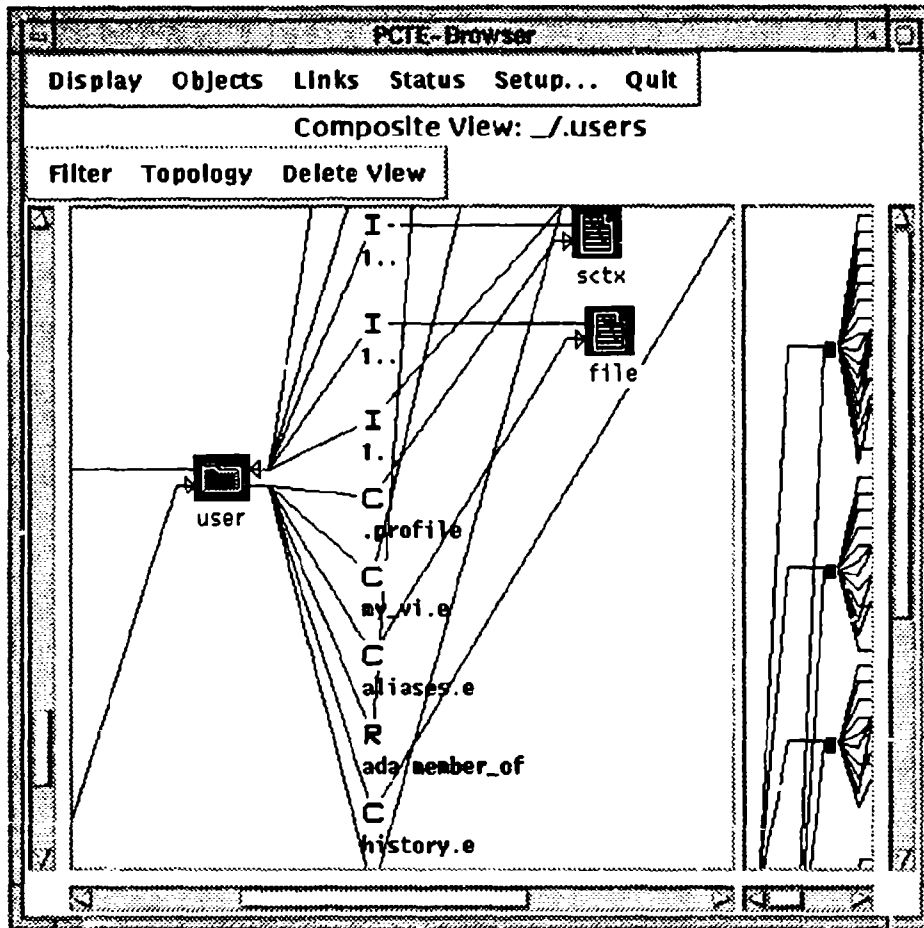


Figure 2: Topology Region Within View

Several PBT commands expect PCTE objects to be identified via their pathnames. Such pathnames can start at either the root of the object base ("-") or the PBT user's home ("~"). Such pathnames can also be relative to the PBT session's *Current Object*. In addition, such pathnames can make use of user-defined *object aliases*. The user may define new aliases (which are local to a PBT session), or modify or delete existing ones. The PBT also supports the saving of such aliases in PCTE file objects, and the loading of aliases from such files.

Since the graph of objects in any particular View may be too large to fit within the window, the PBT supports mechanisms for scrolling around the graph. The PBT also allows the user to attach a *Topology* region to any existing View—a region that contains a miniaturized depiction of the graph, sometimes called the "view from one thousand feet." Figure 2 illustrates the same graph displayed in figure 1, with the Topology region now added to the right portion of the View.

The PBT supports a number of mechanisms by which one can filter out objects and links from the graphs displayed within Views.

Prior to the creation of Views, the user can control which links are to be included in such Views in two ways: via the PCTE *working schema* and via user-specifiable link name patterns.

The PBT allows the user to modify the set of SDSes are included in the PBT session's current working schema. Links whose names are not defined in this working schema will not be displayed in Views. In addition, the PBT allows the user to specify any number of link type names or name patterns; if any such patterns have been specified, then the PBT further limits the displayed links to be those matching such type name patterns. Link type name patterns can include the UNIX shell-like wildcards "*" and "?" (e.g., "*list"). The PBT allows the user to save these patterns in PCTE file objects, and to load previously saved patterns from such files.

For graphs of objects displayed in existing Views, the PBT supports the hiding and reexposing of objects and/or links within these Views: PCTE objects of a specific kind (e.g., all PCTE files or all non-file objects) can be hidden or exposed, as can links of a specific category (e.g., all composition, reference or implicit links). The PBT can also hide or expose objects based upon the names of their object type names, and can hide or expose links based upon their link type names.

The PBT allows the user to tailor some aspects of the PBT behavior, such as:

- whether SDS names are to be used in Views within object and link labels;
- the default maximum depth of composition trees associated with Composite Views;
- whether user requests to delete Views or requests to quit the browser should be confirmed;
- whether to display PCTE exception information when PCTE exceptions are raised.

2 References

The following documents are applicable to the PCTE Browser Tool:

- PBT Version Description Document, Version 1.0, 30 November, 1992.
- Standard ECMA-149, Portable Common Tool Environment (PCTE) Abstract Specification, December, 1990.
- Emeraude V12, release 3, System Administration, January, 1992.
- Emeraude V12, release 3, Environment Guide, January, 1992.
- Emeraude V12, release 3, Tool Catalogue, January, 1992.

- Quercia, Valeria and O'Reilly, Tim, X Window System User's Guide for X11 R3 and R4, Volume 3, 3rd edition, O'Reilly & Associates.
- Fulton, Jim, X Window System, Version 11 Release 4 Release Notes, (supplied with MIT distribution)
- OSF/Motif Programmer's Reference, Revision 1.1, 1991.
- RGB User Manual, STARS-US-020401/002/00, 20 November, 1992.

3 INSTRUCTIONS FOR USE

3.1 PREPARATIONS FOR USE

In order to use the PBT, the following preliminary steps must be taken:

1. The PBT executable must be installed in the environment so that it will be accessible by PCTE users.
2. A PBT-oriented SDS (named `pbt`) must be installed in the environment so that it may be included in the current working schema by the PBT. (NOTE: This SDS is used to implement the action and file object contents viewing mechanisms described in sections 4.8.3 and 4.8.4 described below.)
3. The set of files describing the X Window System resources and bitmaps to be used by the PBT must be installed in the UNIX file system, and the location of these files must be identified via the `XAPPLRESDIR` environment variable.
4. The X Window System, X11R4, must be running on the user's display.
5. The server process for the PCTE object base must be running.
6. The user must be logged into PCTE.

3.1.1 Installing the PBT Executable

In the Emeraude implementation of PCTE, an executable such as PBT can either be installed as a *static context* (type `env-sctx`) within the PCTE object base itself, or can be moved into one of the UNIX directories within the UNIX path in affect during a PCTE session.

3.1.2 Installing the `pbt` SDS

One of the files that is part of the PBT delivery is `pbt.sds`. This file contains the PCTE schema data definition language (DDL) specification of the `pbt` SDS. This SDS defines two link types that are used by the browser:

- *viewer* links — which go from PCTE file objects to PCTE static contexts. The PBT includes a mechanism by which these static contexts can be invoked for the associated file objects. (See section 4.8.3 below for more information.)
- *cction* links — which go from all types of PCTE objects to PCTE static contexts. The PBT includes a mechanism by which these static contexts can be invoked for the associated objects. (See section 4.8.4 below for more information.)

The SDS is installed in the PCTE object base via the `sds_compile` command. See the accompanying VDD for more details on installing the SDS.

3.1.3 Installing the X Resource Files

A number of UNIX files associated with the PBT must be on-line at the time that the PBT is executed:

- **PCTE-Browser**, the “X resource file” associated with the PBT.
- A set of files describing the bitmaps to be used for the various object and link icons.

These files are part of the PBT delivery. See the accompanying VDD for a description of how to install these files.

The PBT is typical of X-oriented tools in that the characteristics of the tool can be set or overridden by the user, rather than being hard-wired into the tool. These characteristics include such things as which bitmaps to use for specific types of object, or what physical display dimensions to use for the various widgets used by the browser. These characteristics (“resource values”) are specified in **PCTE-Browser**, which is read by the browser each time it is started, and therefore, can be customized for each site or for individual users of the PBT. Appendix A contains the color monitor-oriented X resource file supplied with the PBT release—one which could be customized at any local installation of the PBT.

3.1.3.1 Identifying the X Resource Files

As discussed above, the PBT reads its X resource specification, **PCTE-Browser**, each time that the browser is invoked. The browser expects to find the newly installed **PCTE-Browser** file in the UNIX directory identified by the `XAPPLRESDIR` environment variable. The following command line could be added to the `.login` or `.cshrc` file of every potential user of the browser, or else could be executed prior to invoking the PBT:

```
setenv XAPPLRESDIR directory-containing-PCTE-Browser
```

where *directory-containing-PCTE-Browser* is to be replaced by the site-specific absolute UNIX pathname of the directory containing the **PCTE-Browser** file.

NOTE: There are two versions of the PCTE X Resource File delivered with the PBT--one for a color monitor, and one for a monochrome monitor. Make sure that the version of PCTE-Browser in the directory identified by XAPPLRESDIR is appropriate for the type of monitor being used.

3.1.4 Starting the X Window System

The PBT can be executed on any workstation running X11R4. This can be the Sun-supplied implementation of X, OpenWindows, or the X distribution from MIT.

It is beyond the scope of this user manual to describe how to install X, or how to start the X server on a workstation. Consult with your local system administrator for help.

There are a number of "standard" window managers that one may use in an X environment. The PCTE browser has been used successfully under the following such window managers: **mwm** (Motif Window Manager) and **twm** (the official MIT-distributed window manager).

3.1.5 Starting the PCTE Server

Consult the Emeraude V12 System Administration guide for information on how to start the PCTE server.

3.1.6 Logging into PCTE

The "standard" method for logging into PCTE is sufficient for using the PBT. Consult the Emeraude V12 System Administration guide for details.

3.1.7 Executing the PCTE Browser Tool

Once the PBT executable and the pbt SDS have been installed in a PCTE object base, the X server, X window manager and PCTE server are executing, and the XAPPLRESDIR environment variable has been set, the PBT may be invoked.

The invocation can be done from any environment capable of creating PCTE processes, such as from within the **esh** shell.

The PBT recognizes and accepts a number of command line arguments. These arguments allow the user to do any or all of the following:

1. Set any or all of the customizable aspects of PBT behavior. (See section 4.11.)
2. Set the initial working schema to be used by the PBT. (See section 4.7.1.2.)

3. Set the initial value of the Current Object. (See section 4.6.1.2.)
4. Load an initial set of object aliases. (See section 4.1.1.6.)
5. Load an initial set of link type name patterns. (See section 4.5.6.)
6. Create an initial set of Local and/or Composite Views. (See sections 4.3.1.2 and 4.3.2.2.)

Refer to Appendix B for a complete listing of the PBT "help" message, which describes all of the command line arguments.

3.1.7.1 Order of Evaluation of Command Line Arguments

The command line arguments are always processed in the order indicated above, regardless of the order in which these arguments are specified. For example, consider the following PBT invocation which both loads object aliases ("-a") and sets the Current Object ("-c"):

```
PBT -a aliases.e -c ~
```

Even though the arguments are entered with the "load aliases" argument indicated first, the setting of the Current Object (step 3) will actually be performed first. As a consequence, the "Current Object-relative" `aliases.e` pathname is relative to `~`; i.e., the full pathname to the aliases file object is interpreted as `~/aliases.e`.

If the PBT detects any syntax errors in the command line arguments, it simply identifies them (by writing error messages to the PBT's standard output device) and then terminates the program. The following is an example of such a PBT invocation attempt:

```
PBT -a aliases.e -q -c ~
Command line argument 3 ("-q") is invalid.
Use the -h option to display all valid PBT options.
```

However, if there are semantic errors in the command line arguments, the PBT does not terminate. Instead, it processes as many of steps in the command line processing as it can up to, but not including, the step in which the semantic error is detected. It then skips over the remaining steps, with warning messages written to standard output as shown below. The PBT opens up the Initial Browser window (shown in figure 3), thus allowing the user to explicitly recover from whatever semantic errors were detected. The following example illustrates what happens when semantic errors are detected. In this example, the specified object to become the initial Current Object does not exist:

```
PBT -a aliases.e -c _/cuest.usr -l
Error setting object reference: "_/cuest.usr" is not an existing object.
```

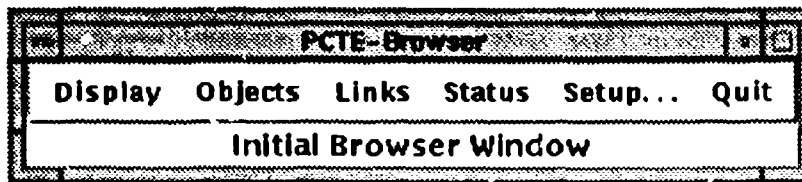


Figure 3: Initial Browser Window

Skipping over the setting of the current object.
Skipping over the loading of the initial object aliases.
Skipping over the creation of the initial Views.

3.2 BROWSER LOOK AND FEEL

The PBT application creates a number of types of Motif-style windows as needed during a PBT session. These windows include:

- an Initial Browser window
- Composite View windows
- Local View windows
- control panels
- dialog boxes
- scrollable text windows
- alert boxes
- confirmation boxes

3.2.1 Initial Browser Window

The Initial Browser window, shown in figure 3, is the first window created by the PBT. This window exists until the first Local or Composite View is created, at which point it is transformed into either a Local or Composite View window.

The Initial Browser window consists of a title bar with the constant title "Initial Browser Window," and a *global* menu bar above the title.

The global menu bar in Initial Browser window contains six command buttons that can be used to invoke global PBT commands—commands which apply to the overall PBT session, rather than to any specific View window. These command buttons are:

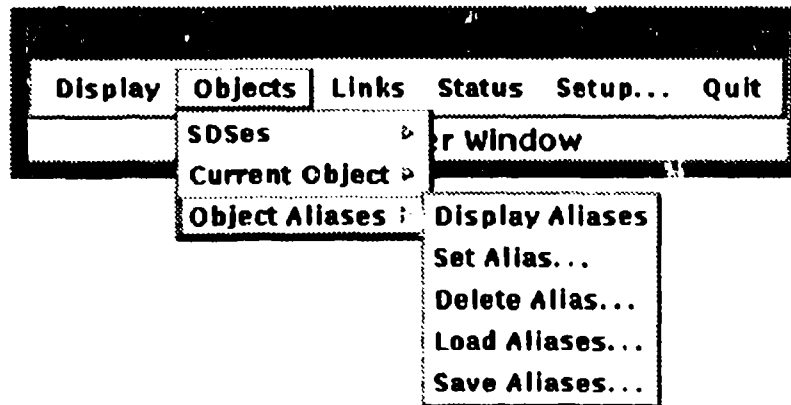


Figure 4: Objects Command Button Menu

- **Display** — used to create new Composite and Local View windows (see sections 4.3.1.1 and 4.3.2.1).
- **Objects** — used to maintain the current working schema (see section 4.7), to maintain the Current Object (see section 4.6), and to maintain the set of object aliases (see section 4.1.1).
- **Links** — used to maintain the set of link type name patterns (see section 4.5).
- **Setup** — used to customize the PBT behavior (see section 4.11).
- **Status** — used to display the most recent PCTE exception information (see section 4.10).
- **Quit** — used to terminate a PBT session.

Three of the global command buttons (**Status**, **Setup** and **Quit**) have only a single PBT command associated with it. To invoke any of these PBT commands, the user simply clicks the left mouse button while the mouse pointer is within the bounds of one of these command buttons.

The other global command buttons (**Display**, **Objects** and **Links**) can each be used to invoke more than one PBT command. Each of these three buttons have pulldown menus that appear when the left mouse button is depressed while the mouse pointer is within the bounds of the command button. To invoke any of these PBT commands, the user selects the desired menu item off one of these pulldown menus; this selection is done by releasing the left mouse button while the mouse pointer is pointing at the menu item.

Figure 4 contains an example of a pulldown menu—the one associated with the **Objects** command button. This particular pulldown menu includes cascaded submenus. Since this is a standard Motif menu, it uses the Motif convention for indicating the presence of a cascaded submenu—a right-pointing arrow head.

Figure 4 also illustrates a PBT menu convention: A number of PBT commands that are invoked via menu selection require additional information to be supplied by the user in order to be complete the command invocation. When selected, these menu items all pop up either a control panel or dialog box in which this additional information can be specified. The PBT indicates that a menu item has an associated popup by including an ellipsis ("...") at the end of the menu item title.

3.2.2 Composite View Windows

Composite View windows display the set of PCTE objects comprising a composite PCTE object beginning at a user-specified PCTE object, as illustrated in figure 5. The PBT can create any number of Composite View windows during a PBT session. These View windows remain in existence until either the PBT session is terminated or until the user explicitly deletes them.

Each Composite View window includes the following:

- a *global* menu bar, containing the same set of global command buttons as in the Initial Browser window;
- a title bar below the global command bar identifying the root of the Composite View;
- a second menu bar (below the title), containing view-specific command buttons;
- a scrollable region at the bottom of the window containing the graph of the composite object and related links.

The lower menu bar contains three command buttons that can be used to invoke view-specific PBT commands—commands which apply to the specific Composite View window, rather than to the PBT session in general. These command buttons are:

- **Filter** — used to hide and/or reveal nodes and arcs withing the View's graph (see section 4.4).
- **Topology** — used to hide and/or reveal the "view from one thousand feet" (see section 4.1.1).
- **Delete View** — used to destroy the Composite View window.

Two of the view-specific command buttons (**Topology** and **Delete View**) have only a single PBT command associated with it. To invoke either of these PBT commands, the user simply clicks the left mouse button while the mouse pointer is within the bounds of the appropriate command button. The third command button (**Filter**) can be used to invoke more than one PBT command; it, therefore, has an associated pulldown menu that appear when the left mouse button is depressed while the mouse pointer is within the bounds of this command button.

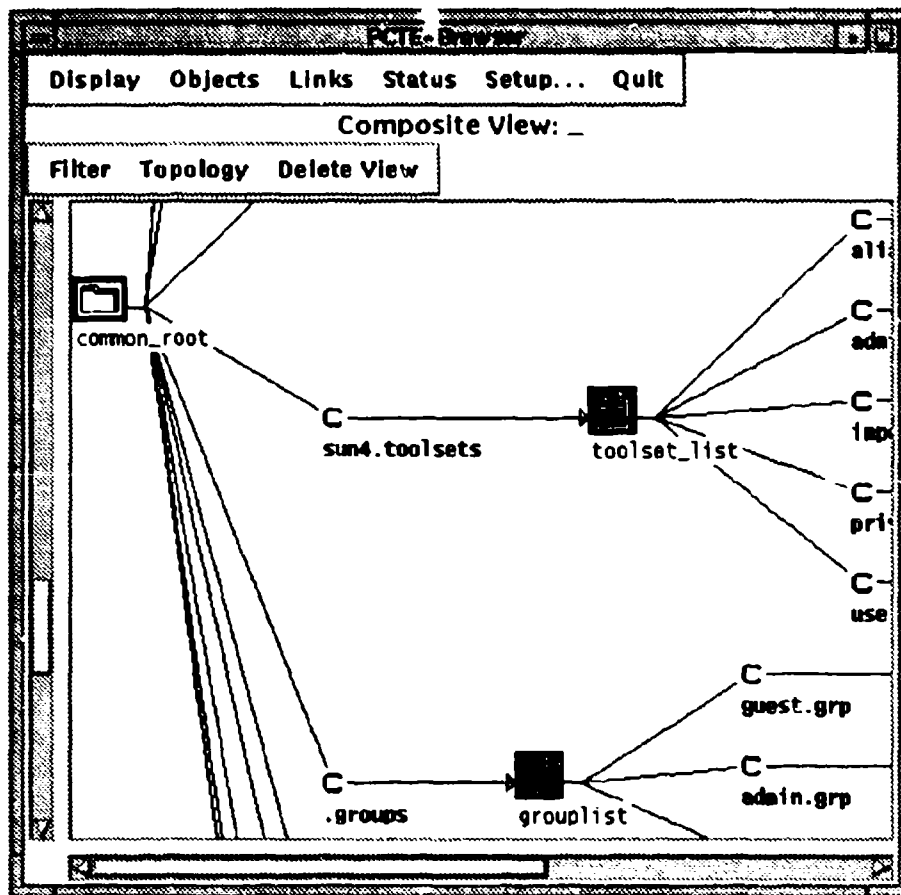


Figure 5: Composite View Window Example

3.2.3 Local View Window

Local View windows display the set of objects that are within a user-specified distance from a user-specified starting object, as illustrated in figure 6. The PBT can create any number of Local View windows during a PBT session. As with Composite View windows, Local View windows remain in existence until the browser session is terminated or until the user explicitly deletes them.

Each Local View window includes same set of components as is found in a Composite View window:

- the global menu bar;
- a title bar below the global command bar identifying the start object of the Local View;
- a second menu bar (below the title), containing the same set view-specific command buttons as found in Composite View windows;
- a scrollable region at the bottom of the window containing the graph of the Local View.

3.2.4 Dialog Boxes

Dialog boxes pop up when a simple text string is needed from the user to complete a requested PBT command. For example, the dialog box illustrated in figure 7 pops up in response to selecting the **Delete Link Type Name Patterns** menu item of the global **Links** command button. The information needed from the user in this case is the pattern to be deleted.

The dialog box includes an editable Motif text field. To insert text within this field, the mouse pointer must first be moved into the rectangle surrounding the text field.

Text will be inserted at the location of the "I-beam" cursor within this field, which appears whenever the mouse pointer is within the bounds of the field. Once text has been inserted into the field, this cursor can be moved by clicking (with the left mouse button) at any point within this text. (Consult the standard Motif text widget documentation for more complete information on moving this cursor.)

Existing text within this field can be replaced by first selecting (i.e., highlighting) some or all of the existing text, then entering new text; this will delete the old, selected text and insert the new text in a single operation. One way to select text is to depress the left mouse pointer at the beginning of text drag the mouse pointer to the other end of the text section. One way of selecting the entire text field is to triple click (with the left mouse button) while pointing anywhere within the text field. Another way to select the entire field is to type *control-slash* while pointing anywhere within the text field. (Consult the standard Motif documentation on the text widget for more complete information on selecting text.)

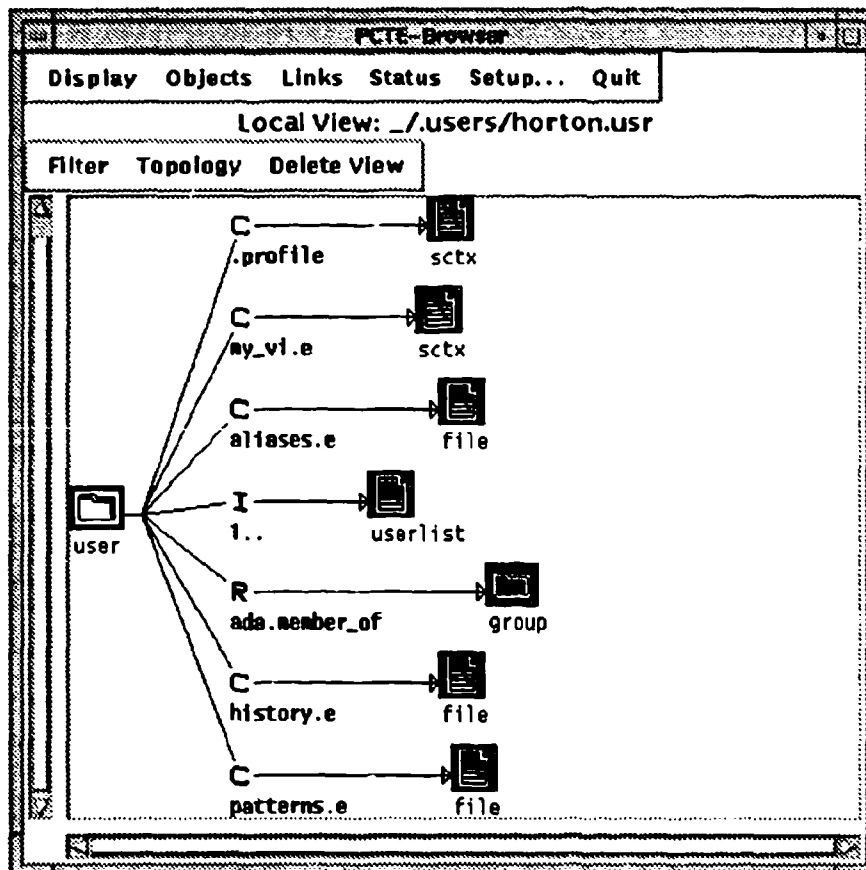


Figure 6: Local View Window Example

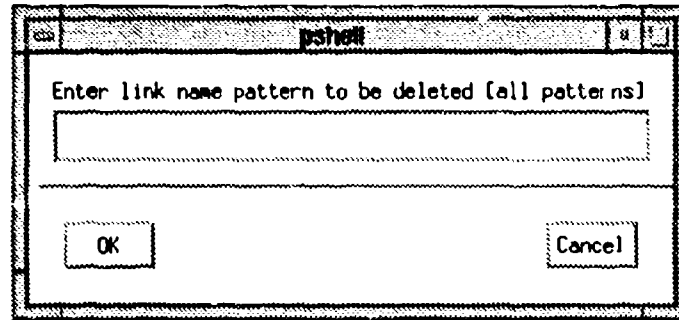


Figure 7: Dialog Box Example

Text to the left and the right of this cursor can be deleted using the *delete* and *back space* keys, respectively. Alternatively, a section of text to be deleted can be selected and then deleted in a single operation via the *delete* key.

A dialog box is deleted as soon as the user clicks (with the left mouse button) in either the OK button (to apply the information and complete the PBT command invocation) or **Cancel** button (to cancel the associated command). NOTE: In general, no other PBT action can be taken until the user responds to the dialog box.

3.2.5 Control Panels

Control Panels are used when information is needed from the user and where a dialog box, with its single text field, is insufficient. For example, the control panel illustrated in figure 8 pops up whenever the user requests the creation of a new Local View. The information needed from the user in this case is the pathname of the object to be used as the Local View's starting point, as well as the maximum distance of other objects from this start object.

Control panels can consist of any or all of the following types of components:

- text fields
- radio buttons
- single selection lists

Text fields within control panels are similar to those found in dialog boxes. (See section 3.2.4 for information on editing such fields.)

Radio boxes are used to select one item from amongst a fixed set of choices. For example, the control panel in figure 19 contains a single radio button component (along with two text fields). The radio button component allows the user to chose which (of three possible choices) sets of link categories to include in a new Composite View. The user selects a radio button

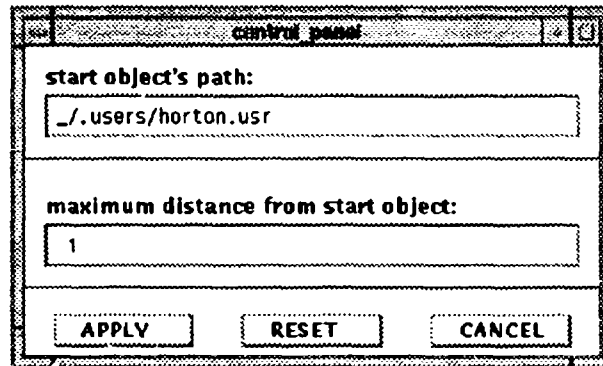


Figure 8: Control Panel Example

choice by clicking (with the left mouse button) in the diamond-shaped icon associated with that choice. NOTE: Radio button components in control panels are designed to ensure that exactly one choice is always selected within these components.

Single selection lists are used to select one item from amongst a dynamic set of choices. For example, the control panel in figure 46 contains a single single selection list component. This particular list component allows the user to chose amongst the available mechanisms for viewing a file object's contents. The user selects an item in such a list by clicking (with the left mouse button) in that item. NOTE: The user need not select *any* item within a single selection list component; in fact, if the user clicks in an already selected list item, that item will be deselected, without selecting any other item.

A control panel is deleted as soon as the user clicks (with the left mouse button) in either the **APPLY** button (to complete the PBT command invocation) or **CANCEL** button (to cancel the associated command). The third button at the bottom of every control panel, **RESET**, can be used at any time prior to clicking in the **APPLY** button; this will reset all of the control panel fields to their values at the time when the control panel first popped up.

3.2.6 Text Windows

Scrollable *text* windows, such as the one illustrated in figure 9, are used by most PBT commands that must display text. Information displayed in these text windows include:

- the current working schema
- the Current Object
- the full set of SDSes in the PCTE metabase
- the current set of link type name patterns
- the current set of object aliases

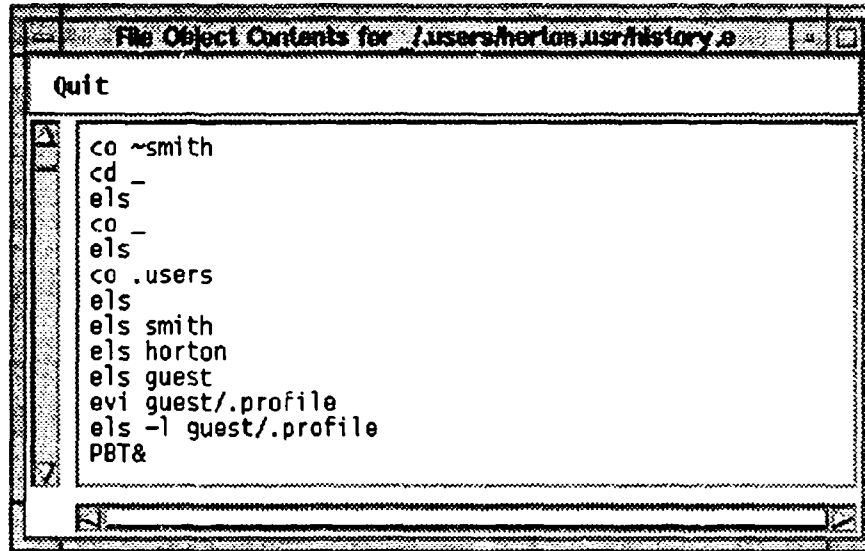


Figure 9: Scrollable Text Window Example

- PCTE file object contents
- the non-key attributes for a particular link
- all the attributes for a particular object
- object pathnames

All PBT text windows contain standard Motif scroll bars. These scroll bars can be used to move about the text window contents when it is too long to be displayed all at once.

The PBT user can simultaneously display any number of text windows. Individual text windows may be deleted by selecting the **Delete the Text Window** menu item (using the left mouse button) from the text window's **Quit** button. The entire set of text windows—and all Local and Composite View windows as well—will be deleted when the global **Quit** command button is selected in any View window.

NOTE: Currently, all text windows created by the PBT are read-only.

3.2.7 Alert Boxes

Alert boxes pop up when unexpected conditions are detected. The alert box illustrated in figure 10 appears when a pathname specified as the root of a new Composite View does not exist.

A alert box will be deleted when the user clicks (with the left mouse button) in the alert box's OK button. Until the user responds, no other PBT action can be taken.

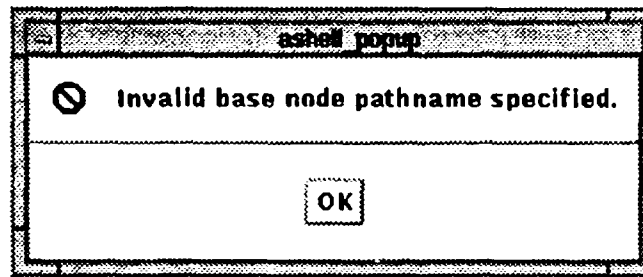


Figure 10: Alert Box Example

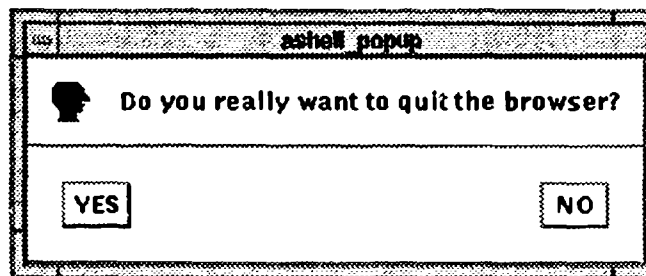


Figure 11: Confirmation Box Example

3.2.8 Confirmation Boxes

Confirmation boxes pop up when a yes/no answer is required from the user to complete a PBT command. The confirmation box shown in figure 11 pops up when the user has requested that the PBT session be terminated; it is used to make sure that user did not invoke the quit operation unintentionally.

A confirmation box will be deleted when the user clicks (with the left mouse button) in the either box's YES or NO button. Until the user responds, no other PBT action can be taken.

3.2.8.1 Use of View's Topology Region

3.2.9 Miscellaneous Window Manager Operations

A number of functions related to the management of browser windows are controlled by the X window manager in use at the time the browser is invoked (e.g., *mwm* or *twm*). These functions include:

- Resizing windows
- Iconifying windows

- Placing windows on the screen and moving them around the screen
- Exposing browser windows covered by other windows
- Hiding browser windows behind other windows

The means by which these operations are accomplished depends upon which window manager is in use. These window managers decorate most browser windows with title bars with "buttons" (mouse-sensitive regions) that can be used to initiate some or all of the above functions. All of the examples illustrated in this document were created using `mwm`, the Motif window manager. Consult the documentation for the specific window manager in use for explanations as to how the title bar buttons are used, and for alternate methods for initiating window manager functions.

4 PCTE Browser Capabilities

The subsections below describe the following PBT capabilities:

- PCTE object identification
- name pattern conventions
- creation/maintenance of View windows
- filtering of objects within View windows
- filtering of links within View windows
- operations on objects
- operations on links
- display of error status information
- customizing the browser
- exiting the browser

4.1 Object Identification

Several PBT functions prompt the user to identify the object in the PCTE object base for which the function is to be applied.

The PBT recognizes standard PCTE pathnames starting either from the root ("`_`") or the user's home ("`~`"). Examples of such pathnames include:

```
~/history.e  
_/.users/guest.usr
```

Relative pathnames are those beginning with a link name, such as: `history.e`. Such pathnames are interpreted within the PBT as relative to the PBT session's *Current Object*. For example, if `_.projects` was the set to be the Current Object earlier in the PBT session, then the previous relative pathname would be interpreted as `_.projects/history.e`. (See section 4.6.1 for information on setting the Current Object.)

4.1.1 Aliases in Object Identification

As a final shorthand method for object identification, the PBT supports the creation and maintenance of a set of aliases whose values are the object pathnames, and allows objects to be identified via the use of these aliases.

Assume for the moment that one such alias already exists, `path_ref`, and assume that the values of this alias is `"_.users/guest.usr"`. Then the following are valid examples of object references:

```
$path_ref  
$path_ref/history.e
```

References to the values of object aliases always begin with a dollar sign ("\$\$") followed immediately by the name of the object alias. The PBT interprets such references by substituting the *\$alias_name* by the current value of the alias. This means that these references are equivalent to:

```
_.users/guest.usr  
_.users/guest.usr/history.e
```

NOTE: The PBT is case-sensitive; i.e., the case used when specifying the name of the alias is significant.

The PBT supports the following aliases-oriented operations:

- Display the current set of object aliases.
- Create new aliases or modify existing ones.
- Delete existing aliases.
- Load alias names and values from existing PCTE files.

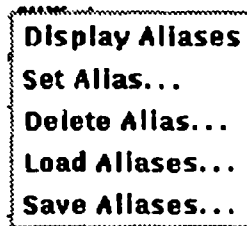


Figure 12: Object Aliases Submenu

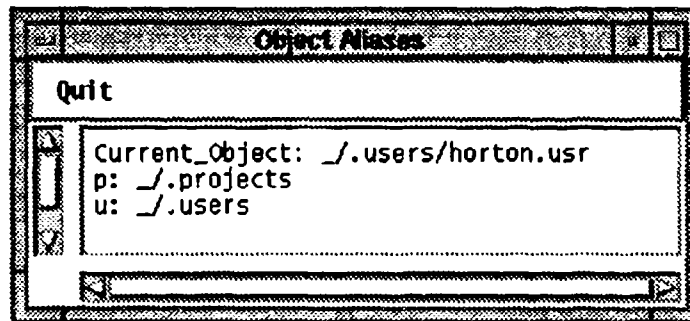


Figure 13: Display Aliases Text Window

- Save the existing set of aliases in new or existing PCTE files.

All of these operations are initiated by selecting menu items off the global **Objects** command button's pulldown menu—more specifically, by selecting menu items from the top-level **Object Aliases** menu item's cascaded submenu (see figure 4). This cascaded submenu is shown in figure 12.

Any of these operations can be invoked at any time during a PBT session. In addition, the PBT supports the capability of optionally loading a set of alias names and values from a single PCTE file object at PBT startup.

4.1.1.1 Displaying Object Aliases

Selecting the **Display Aliases** submenu item pops up a text window that contains the current set of object aliases, as illustrated in figure 13. The aliases are displayed one per line, as in:

Alias Name: Alias Value

Note that **Current Object** is one of the aliases displayed in figure 13. This is a consequence of the fact that the PBT treats the Current Object internally as an object alias, rather than relying upon the underlying PCTE implementation to manage it.

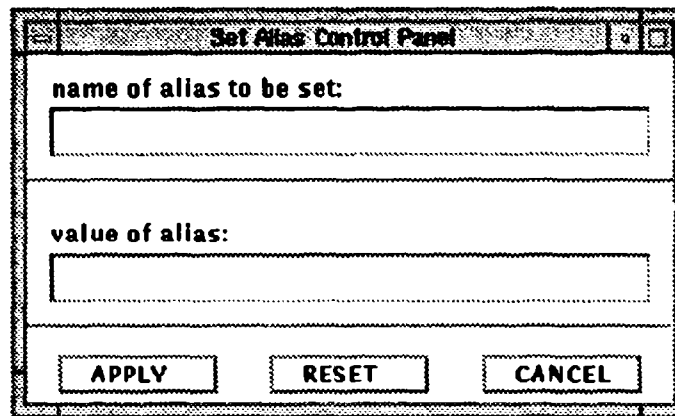


Figure 14: Set Alias Name Control Panel

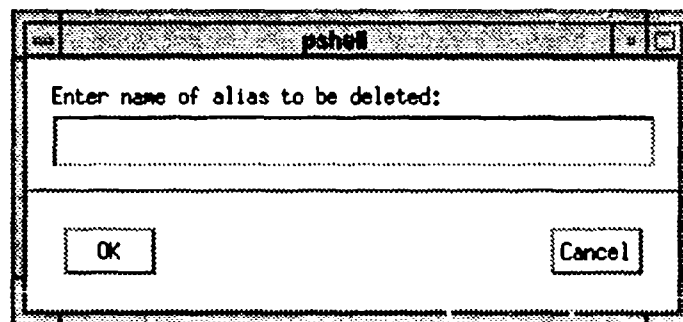


Figure 15: Delete Alias Dialog Box

4.1.1.2 Setting Object Aliases

Selecting on the **Set Alias** menu item causes a control panel shown in figure 14 to pop up. The user can then enter the name of an alias and its associated value (i.e., PCTE pathname). When the user clicks in this Panel's **APPLY** button, the PBT will either replace the previous value of the alias (should one exist), or will add the new alias to the existing set of aliases.

4.1.1.3 Deleting Object Aliases

Clicking on the **Delete Alias** option causes a dialog box shown in figure 15 to pop up. By default, all existing aliases (other than **Current.Object**)—names and values—will be deleted. If the user enters a specific alias name, then only that alias will be deleted.

4.1.1.4 Saving Object Aliases

Clicking on the **Save Aliases** option causes a dialog box shown in figure 16 to pop up. This allows the user to identify the PCTE file in which to save the current set of object aliases.

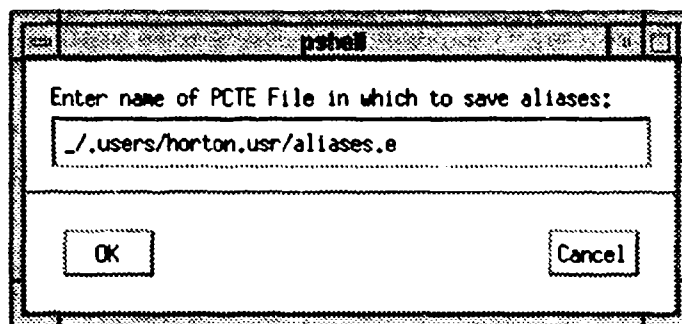


Figure 16: Save Aliases Dialog Box

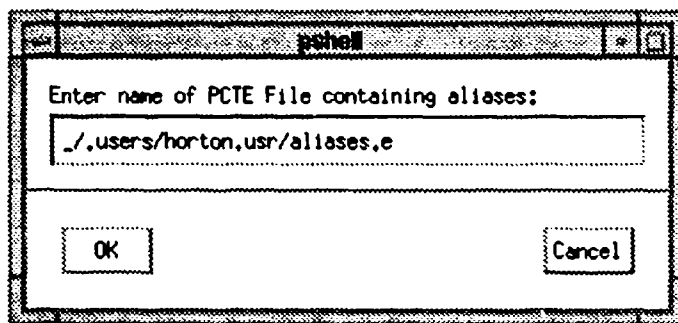


Figure 17: Load Aliases Dialog Box

If this file already exists, its contents will be overwritten by this list of aliases. Otherwise, the PBT will create this file. (NOTE: The one alias that is not saved in this operation is the current value of *Current_Object*.)

The format used for saving the object alias names and values in the file object is exactly the same as that used for displaying them, as shown in figure 13. It is also the same format expected by the *Load Aliases* option discussed below.

4.1.1.5 Dynamic Loading of Object Aliases

Clicking on the *Load Aliases* option causes a dialog box shown in figure 17 to pop up. This allows the user to identify a PCTE file object containing a set of object aliases to be used for referencing purposes. If any aliases already exist at the time a new set is loaded, the new set is merged with the existing set.

4.1.1.6 Loading Object Aliases at FBT Session Startup

By default, the only alias that is initially defined is *Current_Object*. However, the user can use the *-a* command line argument to specify a PCTE file object from which to load an

initial set of aliases. The syntax of the **-a** argument is as follows:

-a *pathname*

NOTE: It is not considered to be an error for more than one instance of the **-a** argument to be specified in a command line argument list; however, all but the last ("rightmost") such argument will be ignored.

4.2 Name Pattern Syntax

Several PBT commands prompt the user to enter a link or object type name, or a name *pattern* that will match one or more such names. The following rules apply whenever such names or patterns are used:

- These names are assumed to be made up of letters, digits, underscores and/or hyphens. All such characters in name patterns match exactly the same character; that is, these names are case-sensitive.
- In addition to the alphanumeric, hyphen and underline characters that can actually comprise these names, two special characters can also be used:
 - "*" matches *any* string of zero or more characters.
 - "?" matches any single character.

If the specified pattern includes a hyphen, then that portion of the pattern to the left of the hyphen will be matched against the name of the SDS to which the type name applies, and that portion of the pattern to the right of the hyphen will be matched against the rest of the type name.

If the specified pattern does not include a hyphen, then any SDS name within the type name will be ignored (automatically matched) and the entire pattern will be matched against the rest of the type name.

4.3 Creation/Maintenance of View Windows

The PBT is able to create two types of Views:

- Composite Views contain a tree of objects formed by the *composition* links from a given root object, optionally including the other types of links emanating from the objects of this tree.
- Local Views contain the set of objects that are within a user-specified distance of a starting object, together with all types of links amongst these objects.

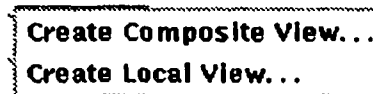


Figure 18: Display Command Button Menu

View creation can be initiated by selecting menu items off the global **Display** command button's pulldown menu, shown in figure 18. It also can be initiated by selecting the "Create View" menu item off any object's popup menu, such as the ones shown in figure 41 and 42.

The subsections below describe how these two types of Views are created, as well as how a Topology region—"the view from one thousand feet"—can be added to or removed from such Views.

4.3.1 Creation of Composite View Windows

Any number of Composite Views can be created dynamically, throughout a PBT session. Any number of such Views can also be created "automatically" at the beginning of a PBT session, based upon user-specified command line arguments. Both methods for creating Composite Views will be discussed below.

4.3.1.1 Dynamic Creation of Composite Views

To dynamically create a Composite View, the user can select the **Create Composite View** option from the global **Display** command button's pulldown menu (shown in figure 18) from the Initial Browser window or any existing Local or Composite View window. Alternatively, one can select the **Create Composite View** option from the popup menu for any object in any existing View window. In either case, selecting the option will cause the Composite View Control Panel shown in figure 19 to appear.

There are three fields in this panel:

- The pathname of the root of the composition tree.
- The category(s) of links that are to included in the View.
- The maximum depth of the composition tree.

If the **Create Composite View** menu item from an object's popup menu was used to initiate the View creation process, then the default value of the root pathname is that object's pathname; otherwise, the pathname of the Current Object will be used as the default. In any case, the user may modify this pathname before applying the Panel.

Any negative value specified as the maximum tree depth is interpreted as an "unlimited" tree, meaning that the entire composition tree will be included in the View.

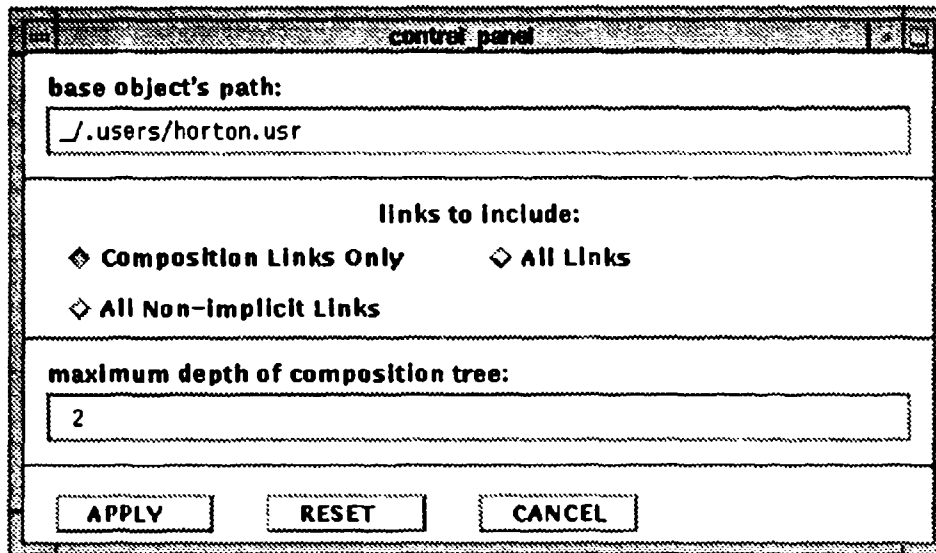


Figure 19: Composite View Control Panel

The default category(s) of links to include in the new View, as well as the default value of the maximum composition tree depth, can be customized either via the **Setup** command button or via a command line argument when starting up the browser. See section 4.11 for more details.

Should the browser detect any problem with the values read from the control panel, such as the specification of a pathname of a non-existent object, then an alert box describing the problem will be displayed. After clicking the Alert's **Ok** button, the user may correct the error and reapply.

NOTE: As is true for all PCTE tools, the PBT is constrained by all the rules governing PCTE access control. This means that objects which the PBT user is not entitled to access will not be included in browser Views.

4.3.1.2 Creating Composite Views at PBT Session Startup

To create one or more Composite Views at PBT startup, the user can specify the **-o** command line argument. The PBT will attempt to create one Composite View for each **-o** argument that the user has specified.

The syntax of the **-o** argument is as follows:

```
-o[link.types][depth] [pathname]
```

The *link.types* component of the **-o** argument can take on any of the following values:

- **c** (i.e., **-oc**) will include composition links only

- **n** (i.e., **-on**) will include all non-implicit links
- **a** (i.e., **-oa**) will include all types of links

If the optional *link_types* is omitted, then the default link types restriction will be used. (Note that this default value can itself be specified as a command line argument. See section 4.11 for more details.)

The *depth* in the **-o** argument can be any integer value. If it is omitted, then the default depth value will be used instead. (This default value can also be specified as a command line argument. See section 4.11 for more details.)

The *pathname* in the **-o** argument specifies the root of the composition tree. If omitted, then the initial value of the Current Object will be used.

The following example illustrates how this command line argument can be used to create two Composite Views:

```
PBT -on3 _/.users/guest.usr -o
```

The first View explicitly uses *_.users/guest.usr* as the root of the tree and 3 as the maximum tree depth, and explicitly includes all non-implicit links.

The second View uses the “built-in” default values for the root pathname, depth and link types to be included: *_, 2 and composition links only*, respectively.

Note that the default link type restriction, the default depth value, and the initial value of Current Object can themselves be specified in other command line arguments. If any or all of these default values are specified in the command line, then these new values will be used for initial Composite View creation, regardless of the order of the command line arguments. To illustrate this, consider the following example:

```
PBT -o -e3 -c _/.users/guest.usr -kn
```

In this example, all of the “built-in” default values affecting Composite View creation are modified: the default depth value is changed to 3, the initial Current Object value is changed to be *_.users/guest.usr*, and the default link kinds to include in Composite Views is changed to be *all non-implicit links*. This means that the one Composite View to be created at PBT startup will use these new defaults, even though the **-o** argument precedes the other command line arguments that specify these default changes.

4.3.2 Creation of Local View Windows

As with Composite Views, any number of Local Views can also be created dynamically, throughout a PBT session. In addition, any number of such Views can also be created

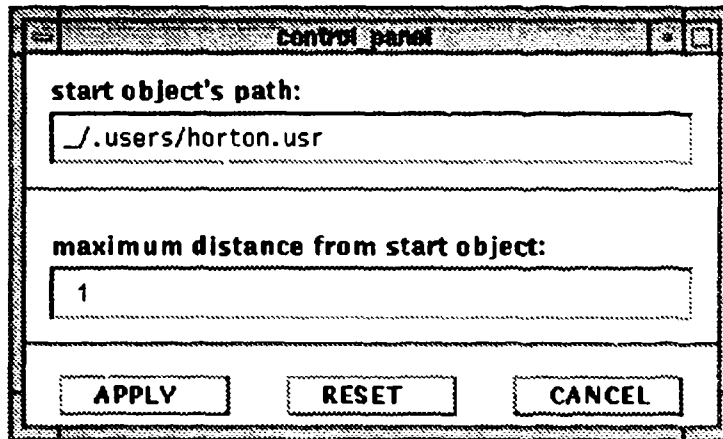


Figure 20: Local View Control Panel

"automatically" at the beginning of a PBT session, based upon user-specified command line arguments. Both methods for creating Local Views will be discussed below.

4.3.2.1 Dynamic Creation of Local Views

To dynamically create a Local View, the user can select the **Create Local View** option from the global **Display** command button's pulldown menu (shown in figure 18). Alternatively, one can select the **Create Local View** option from the popup menu for any object in any existing View window. In either case, selecting the option will cause the control panel shown in figure 20 to appear.

There are two fields in this panel:

- The pathname of the starting object for the Local View.
- The maximum distance from the starting object.

If the **Create Local View** menu item from an object's popup menu was used to initiate the View creation process, then the default value of the start object's pathname is that of the selected object; otherwise, the pathname of the Current Object will be used as the default. In any case, the user may modify this pathname before applying the panel.

The default distance from the start object can be customized either via the **Setup** command button or via a command line argument when starting up the browser. See section 4.11 for more details.

Should the browser detect any problem with the values read from the control panel, such as the specification of a pathname for a non-existent object, then an alert box describing the

problem will be displayed. After clicking the Alert's **Ok** button, the user may correct the error and reapply the panel.

4.3.2.2 Creating Local Views at PBT Session Startup

To create one or more Local Views at PBT startup, the user can specify the `-l` command line argument. The PBT will attempt to create one Local View for each `-l` argument that the user has specified.

The syntax of the `-l` argument is as follows:

```
-l[distance] [pathname]
```

The *distance* in the `-l` argument can be any non-negative integer value. If it is omitted, then the default distance value will be used instead. (Note that this default value can also be specified as a command line argument. See section 4.11 for more details.)

The *pathname* in the `-l` argument specifies the starting object of the Local View. If omitted, then the initial value of the Current Object will be used.

The following example illustrates how this command line argument can be used to create two Local Views:

```
PBT -l1 ./users/guest.usr -l
```

The first View explicitly uses `./users/guest.usr` as the start of the View and `1` as the maximum distance from this start object.

The second View uses the "built-in" default values for the start pathname and maximum distance: `_` and `2`, respectively.

Note that the default link type restriction, the default distance value, and the initial value of Current Object can themselves be specified in other command line arguments. If any or all of these default values are specified in the command line, then these values will be used for initial Local View creation, regardless of the order of the command line arguments. To illustrate this, consider the following example:

```
PBT -l -i1 -c ./users/guest.usr
```

In this example, all of the "built-in" default values affecting Local View creation are modified: the default distance value is changed to `1` and the initial Current Object value is changed to be `./users/guest.usr`. This means that the one Local View to be created at PBT startup will use these new defaults, even though the `-l` argument precedes the other command line arguments that specify these default changes.

4.3.3 Creation of Topology Regions in View Windows

One way to move about a large graph within a View window is via the scroll bars that are present in every View window. A second method for moving the section of the graph being displayed involves the use of the Topology region of the View, as illustrated above in figure 2. This region is not part of the View window when first created, but can be added to the View at any time simply by clicking in the View's **Topology** command button; if the Topology region is already being displayed in the View, then clicking in the **Topology** button will cause the region to be hidden.

The darkened area within the Topology region identifies the portion of the whole graph that is being displayed in the left ("Main") region of the View window. If the left mouse button is clicked anywhere in the Topology region, the section of the graph shown in the Main graph region will be moved such that the location in the graph at the place clicked will be displayed, centered in the Main region if possible. As a side-effect, the darkened area within the Topology region will move as well to reflect what is displayed in the Main region.

Note that the Topology region may not be big enough to display the entire graph, even in miniaturized form, as is shown in figure 2. In such cases, the Topology region's own scroll bars may be used to scroll around the Topology region.

4.4 Filtering Objects and Links Within View Windows

Once a graph of objects and links has been displayed in a Local or Composite View window, the PBT allows the user to specify a set of objects and/or links to be filtered out, i.e., made invisible. Selected objects can either be specified by basic kind (e.g., file or non-file object), or by their object type names—the names used to label the objects within the Views; such filtering of objects also hides those links originating and/or terminating at the filtered out objects. In addition, the user can specify a set of links, either by category (e.g., *composition* or *reference*) or by link type name, and the PBT will redraw the graph without including such links.

This filtering of the objects and/or links in an existing View is initiated by selecting the View's **Filter** command button. This causes the **Filter** pulldown menu shown in figure 21 to appear. After this, the type of hiding/revealing that takes place depends upon the command option (i.e., menu item) selected.

4.4.1 Hiding/Revealing Objects Based Upon Object Kind

The **Suppress Objects By Object Kind** and **Unsuppress Objects By Object Kind** menu items both have the same cascaded submenu, shown in figure 22. By selecting one of the menu items in these submenus, the user can choose to hide (or reveal) any single kind of object, or to hide (or reveal) selected kinds of objects. (In anticipation of a conforming ECMA PCTE environment, Process is one kind of object that is included, even though, in

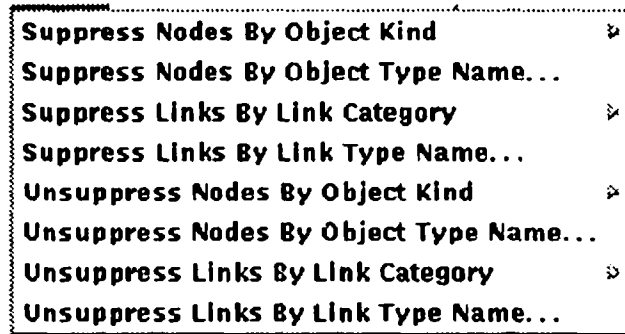


Figure 21: Filter Command Button Menu

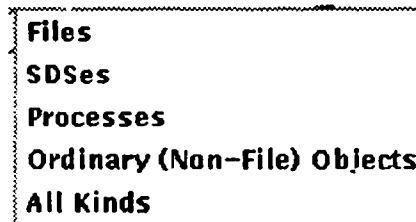


Figure 22: [Un]Suppress by Object Kind Submenu

the PBT implementation on top of Emeraude V12, such objects do not yet exist.)

4.4.2 Hiding/Revealing Links Based Upon Link Category

The **Suppress Objects By Link Category** and **Unsuppress Objects By Link Category** menu items both have the same cascaded submenu, shown in figure 23. By selecting one of the menu items in these submenus, the user can then choose to hide (or reveal) selected categories of links. (In anticipation of a conforming ECMA PCTE environment, *existence* and *designation* link categories are included, even though, in the PBT implementation on top of Emeraude V12, such link categories do not yet exist.)

NOTE: The removal of links from the graph does not affect the set of objects being displayed; i.e., objects will be displayed even if they are totally disconnected from the rest of the objects in the graph.

4.4.3 Hiding/Revealing Objects Based Upon Object Type Name

If the **Suppress Objects By Object Type Name** menu option is selected, the PBT will pop up a dialog box shown in figure 24. The user can then enter the object type name to be hidden, or an object type name pattern describing a set of such names (as defined in section 4.2 above) to be hidden as a group. These names are the ones used to label the various objects within View windows, e.g., `common.root`, `toolset`, `group`, `user`, etc. As with

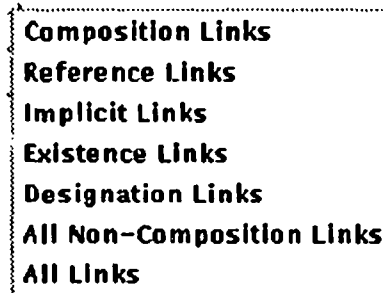


Figure 23: [Un]Suppress by Link Category Submenu

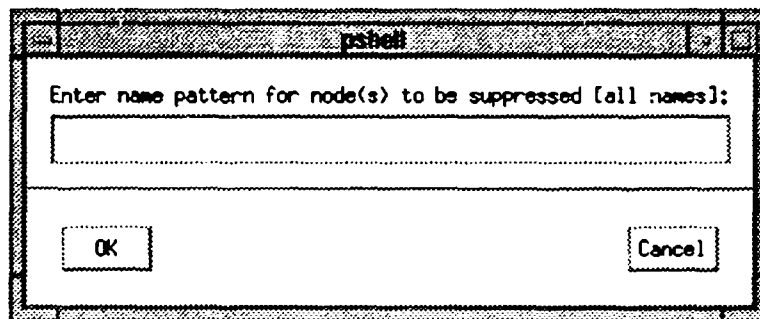


Figure 24: Suppress Object Name Dialog Box

the filtering of objects by more general object kinds, discussed in section 4.4.1 above, links emanating or terminating at hidden objects will also be hidden.

Similarly, if the **Unsuppress Objects By Object Type Name** menu option is selected, the PBT will pop up a dialog box shown in figure 25. The user can then enter the name of object type to be revealed, or a name pattern describing a set of such names.

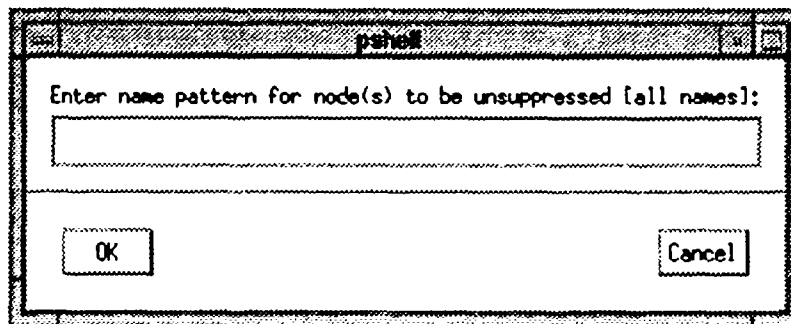


Figure 25: Unsuppress Object Name Dialog Box

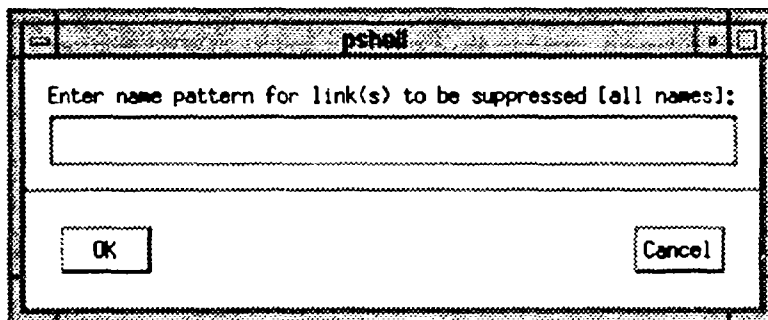


Figure 26: Suppress Link Name Dialog Box

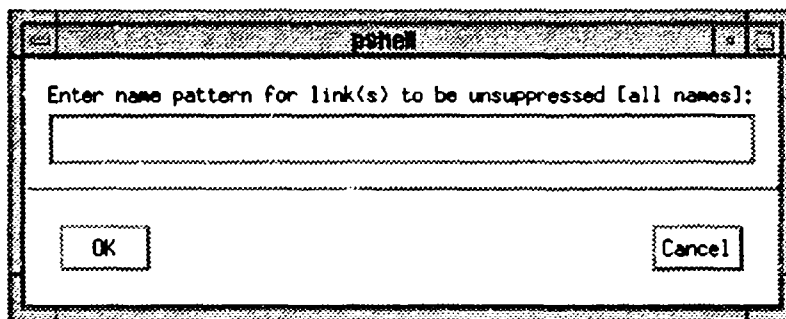


Figure 27: Unsuppress Link Name Dialog Box

4.4.4 Hiding/Revealing Links Based Upon Link Type Name

If the **Suppress Links By Link Type Name** menu option is selected, the PBT will pop up a dialog box shown in figure 26. The user can then enter the link type name to be hidden, or a link type name pattern describing a set of such names (as defined in section 4.2 above) to be hidden as a group.

Similarly, if the **Unsuppress Links By Link Type Name** menu option is selected, the PBT will pop up a dialog box shown in figure 27. The user can then enter the name of link to be revealed, or a name pattern describing a set of such names.

4.5 A Priori Filtering of Links

The previous section described how one can remove objects and/or links from an existing graph in a View in order to focus on what may be most important to a user at any given moment. This section discusses a PBT capability that allows the user to prune objects and links from a graph as the graph is being constructed.

The user can (optionally) create and maintain a set of link type name patterns (e.g., ***list**). If such a list of patterns has been specified at the time that a Local or Composite View is constructed, the PBT will ignore any link that does not match at least one of the specified

patterns when it creates new Views.

Any existing, non-empty list of link name patterns affects the construction of Composite Views in the following manner: If any of the composition links that make up the tree of objects rooted at the specified starting object do not match any of the listed patterns, then the entire subtree of objects "below" that unmatched composition link will be omitted from the constructed graph. In addition, any other categories of links emanating from any objects in the "reachable" portion of the composition tree, will, likewise be excluded from the constructed graph if they don't match at least one of the specified patterns.

Any non-empty list of link type name patterns affects the construction of Local Views in the following manner: Objects will be included in the constructed graph only if they are within the specified distance from the start object traversing only those links whose type names match at least one of the specified patterns. Also, the only links included in the constructed graph will be those matching one of the specified patterns.

It is important to note that the filtering/unfiltering of objects and links discussed in the previous section apply only to those objects and links that are included in the initially constructed View. That is, it is not possible to add objects and/or links that were excluded from the graph initially without modifying (or eliminating) the list of link name patterns and "recomputing" the graph by creating a brand new View.

The PBT supports the following link type name-oriented operations:

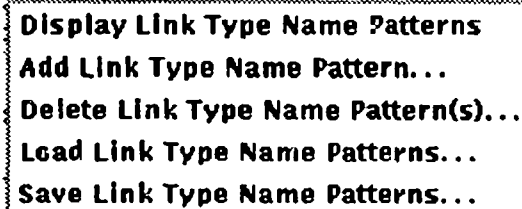
- Displaying the current set of link type name patterns.
- Specifying new patterns.
- Deleting existing patterns.
- Loading patterns from existing PCTE file objects.
- Saving the existing set of link name patterns in PCTE file objects.

All of these operations are initiated by selecting menu items off the global **Links** command button's pulldown menu, shown in figure 28.

Any of these operations can be invoked at any time during a PBT session. In addition, the PBT supports the capability of optionally loading an initial set of link name patterns from a single PCTE file object at PBT startup.

4.5.1 Displaying Link Type Name Patterns

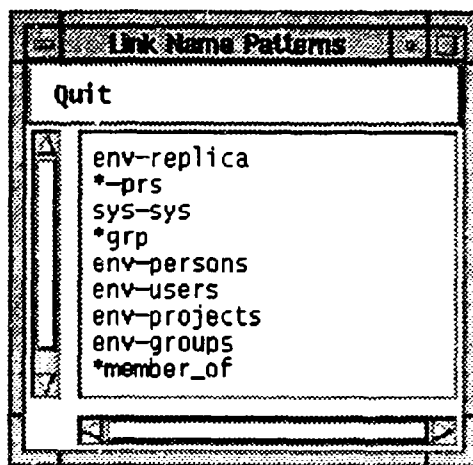
Selecting the **Display Link Type Name Patterns** menu item pops up a text window that contains the current list of name patterns, as illustrated in figure 29.



A menu box with a dotted border containing five items:

- Display Link Type Name Patterns
- Add Link Type Name Pattern...
- Delete Link Type Name Pattern(s)...
- Load Link Type Name Patterns...
- Save Link Type Name Patterns...

Figure 28: Links Menu



A window titled "Link Name Patterns" with a "Quit" button at the top. Below the button is a text area containing a list of patterns:

- env-replica
- *-prs
- sys-sys
- *grp
- env-persons
- env-users
- env-projects
- env-groups
- *member_of

At the bottom of the window is a horizontal scrollbar.

Figure 29: Display Patterns Text Window

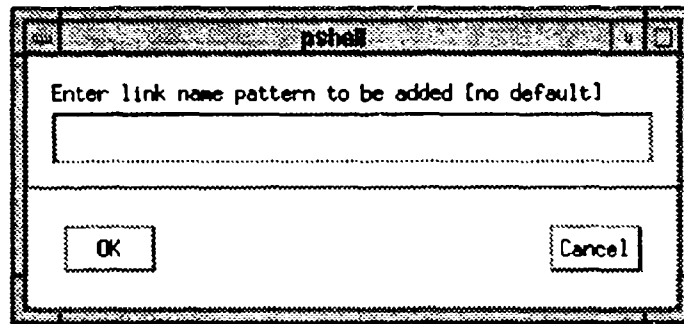


Figure 30: Add Pattern Dialog Box

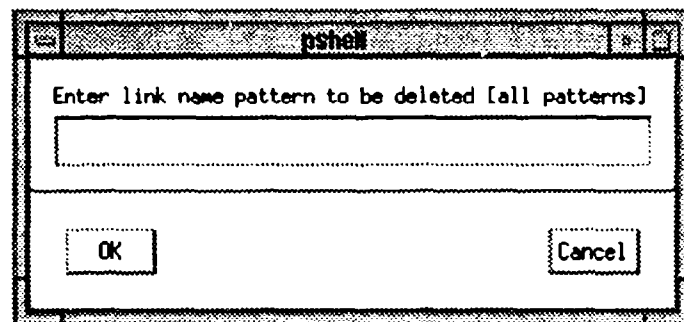


Figure 31: Delete Pattern(s) Dialog Box

4.5.2 Adding Link Type Name Patterns

Selecting the **Add Link Type Name Pattern** menu item causes a dialog box to pop up, as shown in figure 30. The user can then enter a new link name pattern to be added to the current set of such patterns.

4.5.3 Deleting Link Type Name Patterns

Selecting the **Delete Link Type Name Pattern(s)** menu item causes a dialog box to pop up, as shown in figure 31. By default, all existing patterns will be deleted. This means that there will be no more pruning of links and objects based upon pattern matching in subsequently constructed Views—at least not until another list of patterns is constructed. If the user enters a specific link name pattern, then only that pattern will be deleted.

4.5.4 Saving Link Type Name Patterns

Selecting the **Save Link Type Name Patterns** menu item causes a dialog box shown in figure 32 to pop up. This allows the user to identify the PCTE file object in which to save the current set of link name patterns. If this file object already exists, its contents will be overwritten by this list of patterns. Otherwise, the PBT will create this file object.

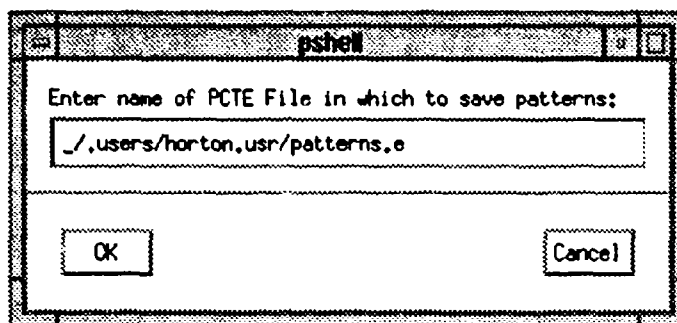


Figure 32: Save Patterns Dialog Box

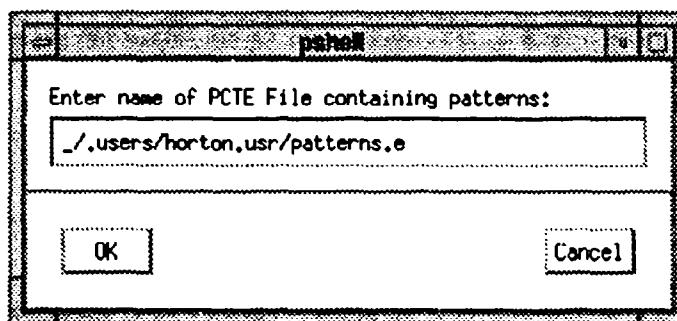


Figure 33: Load Patterns Dialog Box

4.5.5 Loading Link Type Name Patterns

Selecting the **Load Link Type Name Patterns** menu item causes a dialog box shown in figure 33 to pop up. This allows the user to identify a PCTE file object containing a set of patterns to be used for filtering purposes. If any patterns already exist at the time a new set is loaded, the new set is merged with the existing set.

4.5.6 Loading Link Type Name Patterns at PBT Session Startup

By default, there are no link name patterns defined when the PBT session is started. However, the `-t` command line argument can be used to specify a PCTE file object from which to load an initial set of patterns. The syntax of the `-t` argument is as follows:

`-t pathname`

NOTE: It is not considered to be an error for more than one instance of the `-t` argument to be specified in a command line argument list; however, all but the last ("rightmost") such argument will be ignored.

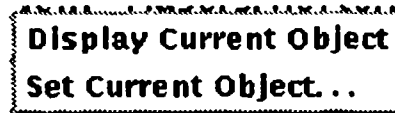


Figure 34: Current Object Menu

4.6 Current Object-Oriented Operations

Current Object-oriented operations include:

- Displaying the PBT session's Current Object.
- Identifying a PCTE object by pathname as the new Current Object.
- Setting a particular object within an existing View to be the Current Object via point-and-click mouse operations.

The first two of the above operations are initiated by selecting menu items off the global **Objects** command button's pulldown menu—more specifically, by selecting menu items from the top-level **Current Object** menu item's cascaded submenu (see figure 4). This cascaded submenu is shown in figure 34.

The third and final Current Object-oriented operation—making a specific object in a specific View into the Current Object—is invoked by selecting the **Set Current Object** menu item off the pop up menu for any object in any View. (See figure 42 for an example of such a menu.)

4.6.1 Setting the Current Object

As noted in section 4.1 above, objects can be identified via Current Object-relative pathnames. In order to make such a capability most useful to the PBT user, the PBT allows the user to dynamically change the designated Current Object at any point within the PBT session. The PBT also supports the setting of the Current Object as part of the PBT startup processing, via command line arguments.

4.6.1.1 Dynamic Setting of the Current Object

To set the Current Object to be any object shown in any existing View window, the user simply selects the object's icon and selects the **Set Current Object** menu item in the popped up menu.

The alternative method for Current Object is to select the **Set Current Object** submenu item from the global **Objects** button's pulldown menu. This will pop up the dialog box

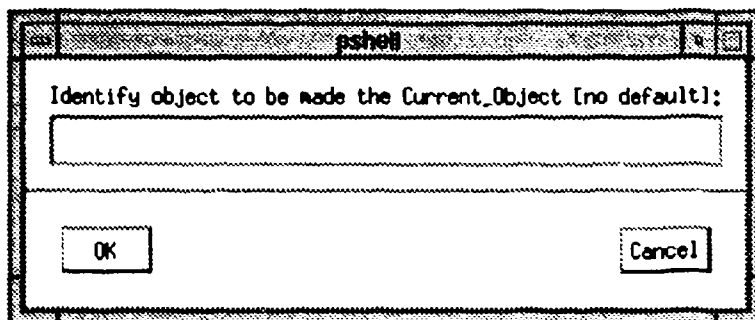


Figure 35: Set the Current Object Dialog Box

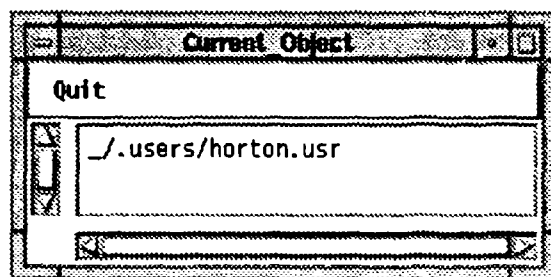


Figure 36: Current Object Pathname Text Window

shown in figure 35, at which point the user types in the pathname identifying the object to be made *current*.

4.6.1.2 Setting of the Current Object at PBT Session Startup

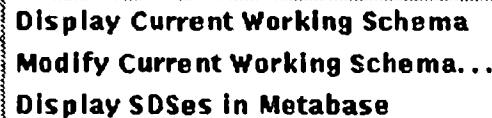
By default, the initial Current Object value is `"_"`. To set the Current Object at PBT startup to some other object, the user can use the `-c` command line argument, the syntax of which is as follows:

`-c pathname`

NOTE: It is not considered to be an error for more than one instance of the `-c` argument to be specified in a command line argument list; however, all but the last ("rightmost") such argument will be ignored.

4.6.2 Displaying the Current Object

In order to display the pathname associated with the Current Object, the user selects the **Display Current Object** submenu item from the global **Objects** button's pulldown menu. This pops up a text window, as illustrated in figure 36.



Display Current Working Schema
Modify Current Working Schema...
Display SDSes in Metabase

Figure 37: SDSes Submenu

4.7 Current Working Schema-Oriented Operations

The schema-oriented operations supported by the PBT include the following:

- Setting the current working schema.
- Displaying the current working schema.
- Displaying the entire list of SDSes in the PCTE object base.

All of these operations are initiated by selecting menu items off the global **Objects** command button's pulldown menu—more specifically, by selecting menu items from the top-level **SDSes** menu item's cascaded submenu (see figure 4). This cascaded submenu is shown in figure 37.

4.7.1 Setting the Current Working Schema

The PBT allows the user to dynamically modify the current working schema at any point within the PBT session. The PBT also supports the setting of this working schema as part of the PBT startup processing, via command line arguments or via a UNIX environment variable.

4.7.1.1 Dynamic Setting of the Current Working Schema

To modify the working schema during a PBT session, the user selects the **Modify Current Working Schema** item from the **Objects** command button's pulldown menu. This pops up the dialog box, as illustrated in figure 38, with the current set of adopted SDSes as the initial value of the list of SDSes. The user can then add to and/or edit this list before ending the dialog.

Should there be any problems with this list, such as the specification of an unknown SDS name, then an appropriate alert box will pop up and the working schema will be unchanged.

NOTE: The PBT requires the working schema to include the **env** and **pbt** SDSes. Therefore, the PBT automatically adds these two SDSes to the end of any user-specified list that omits them.

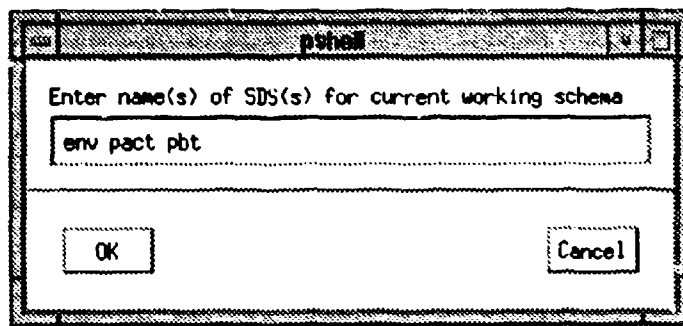


Figure 38: Set PCTE Working Schema Dialog Box

4.7.1.2 Setting the Current Working Schema at PBT Session Startup

By default, the working schema at the start of a PBT session is the following:

```
env pact pbt
```

To set the Current Object at PBT startup to some other object, the user can either use the `-s` command line argument or the `PBT_WS` environment variable.

The syntax of the `-s` argument is as follows:

```
-s working_schema
```

NOTE: It is not considered to be an error for more than one instance of the `-s` argument to be specified in a command line argument list; however, all but the last ("rightmost") such argument will be ignored. Also note that if there are more than one SDSes in the working schema list, then the working schema argument should be quoted, as in the following example:

```
-s "c_prog pact dsmcs"
```

If the `PBT_WS` environment variable has been set prior to invoking the PBT, then its value is assumed to be a list of SDS names to adopt as the initial working schema at the start of the PBT session.

The following is an example of an `esh` session fragment in which an initial value for `PBT_WS` has been set:

```
esh$ PBT_WS="c_prog pact env dsmcs vmcs"
esh$ echo $PBT_WS
c_prog pact env dsmcs vmcs
esh$ export PBT_WS
esh$ PBT &
<16146>
esh$
```

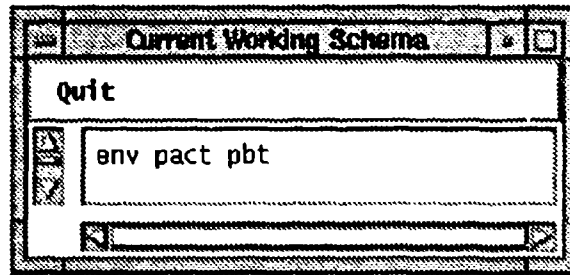


Figure 39: Current Working Schema Text Window

NOTE: The `-s` command line argument takes precedence over the `PBT_WS` environment variable.

4.7.2 Displaying the Current Working Schema

To display the current working schema, the user selects the **Display Current Working Schema** item from the **Objects** command button's pulldown menu. This pops up a text window containing the current set of adopted SDSes, as illustrated in figure 39.

4.7.3 Displaying SDSes in Metabase

To display the list of all of the SDSes in the metabase—or at least all of the SDSes for which the PBT has read access rights—the user selects the **Display SDSes in Metabase** item from the **Objects** command button's pulldown menu. This pops up a text window containing the list of such SDSes, as illustrated in figure 40.

4.8 Object-Oriented Operations

The PBT also supports a number of operations that can be applied to objects found within View windows. For such operations, the user selects the object not by typing in its pathname, but rather by depressing the left mouse button on the object's icon. Such operations include:

- Displaying the full PCTE pathname for the selected object.
- Displaying the object's attributes.
- Setting the PBT session's *current object* to be the selected object.
- Generating a new Composite View rooted at the object.
- Generating a new Local View starting at the object.
- Invoking an action for the selected object.

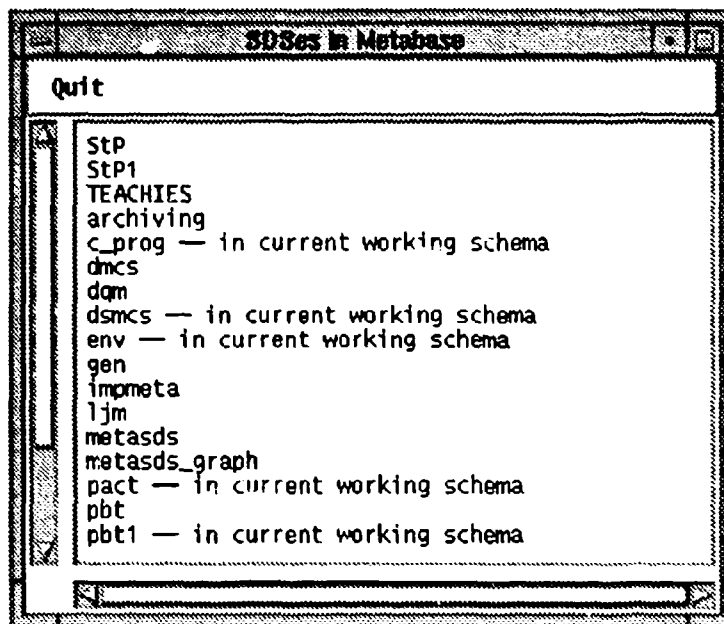


Figure 40: SDSes in Metabase Text Window

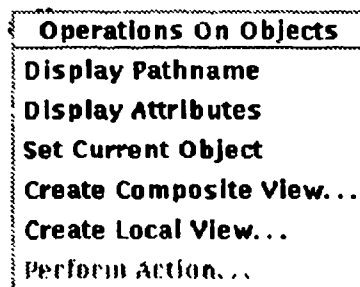


Figure 41: Non-File Object Menu

- Displaying the object's contents.

The first six of these operations can be applied to any type of object found in every View window. The last operation can only be applied to PCTE files. To control which operations can be applied to particular types of objects, the PBT pops up a menu showing exactly the set of operations meaningful to the selected type of object. That is, figure 42 shows the menu popped up for file objects, while figure 41 shows the menu popped up for all other (non-file) objects.

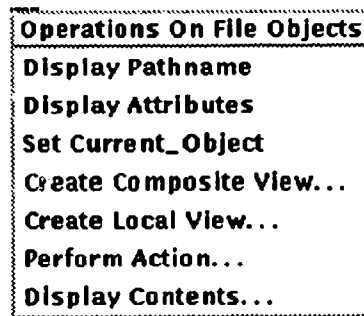


Figure 42: File Object Menu

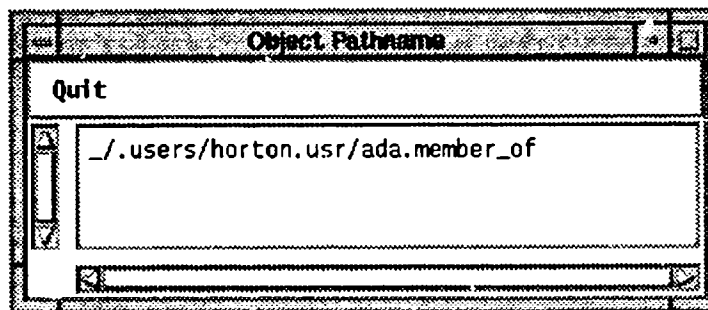


Figure 43: Object Pathname Text Window

4.8.1 Displaying the Pathname

When the **Display Pathname** menu item is selected, the PBT will pop up a text window with the pathname displayed in it, as illustrated in figure 43.

4.8.2 Displaying Object Attributes

When the **Display Attributes** menu item is selected, the PBT pops up a text window with all the attributes in it, as illustrated in figure 44.

4.8.3 Displaying PCTE File Contents

Any PCTE file object containing pure ASCII text can be displayed using the PBT's "built-in" file contents viewer—a standard Motif text window. In addition, if there is a **pbt-viewer** link from a file object to a PCTE *static context*, then that static context can be used as a specialized display program for that file object.

When the **Display Contents** menu item is selected for an ASCII file object which has no **pbt-viewer** emanating from it, the PBT immediately pops up a text window with the contents in it, as illustrated in figure 45. If the object does not have any contents or if the

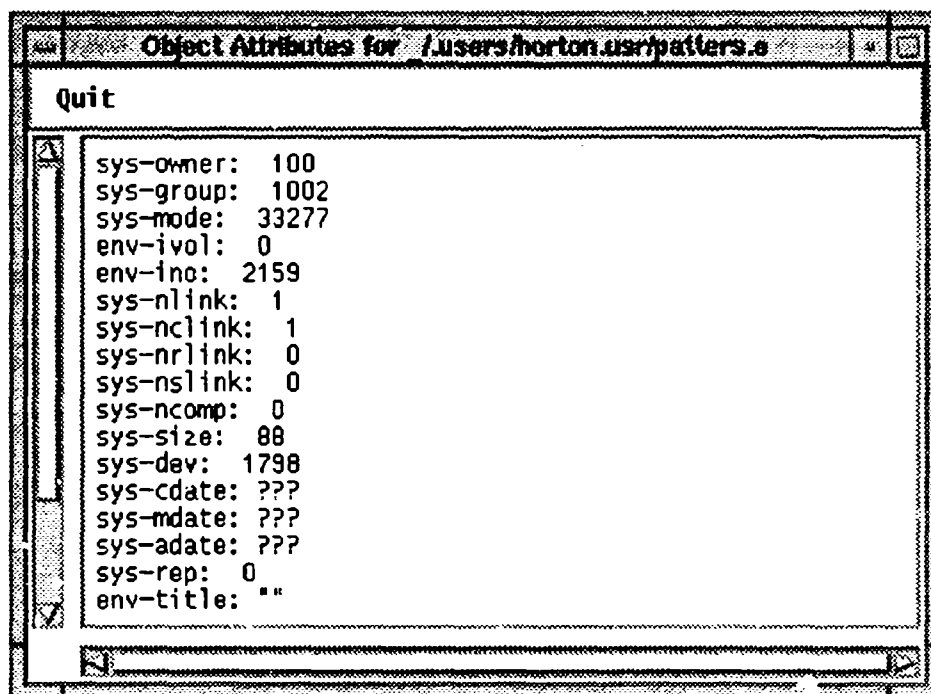


Figure 44: Object Attribute Values Text Window

contents are not pure ASCII text, then the PBT instead pops up an appropriate alert box indicating the problem.

When the **Display Contents** menu item is selected for an file object which does have at least one **pbt-viewer** link emanating from it, the PBT pops up a Viewer Selection Control Panel, such as the one shown in figure 46. This allows the user to choose between the standard PBT viewer (the Motif text window) and any of the specialized viewer tools. NOTE: The key values for the **pbt-viewer** links are assumed to be meaningful enough to identify the viewer program; therefore, these key values are used as the selection list line contents in the popup control panel.

The mechanism for invoking the viewer tool is as follows:

If the "tool" is an ASCII text file, then it is assumed to be an **esh** script, and the shell will be invoked to interpret the script. Otherwise, the tool will be invoked directly.

The pathname of the **pbt-viewer** link's source object is passed to the invoked program.

If the **pbt-viewer** link has a non-null value for the (string) **pre_args** attribute, then this attribute value will be passed to the invoked program as the argument preceding the source object's pathname.

If the **pbt-viewer** link has a non-null value for the (string) **post_args** attribute, then this attribute value will be passed to the invoked program as the argument following the source

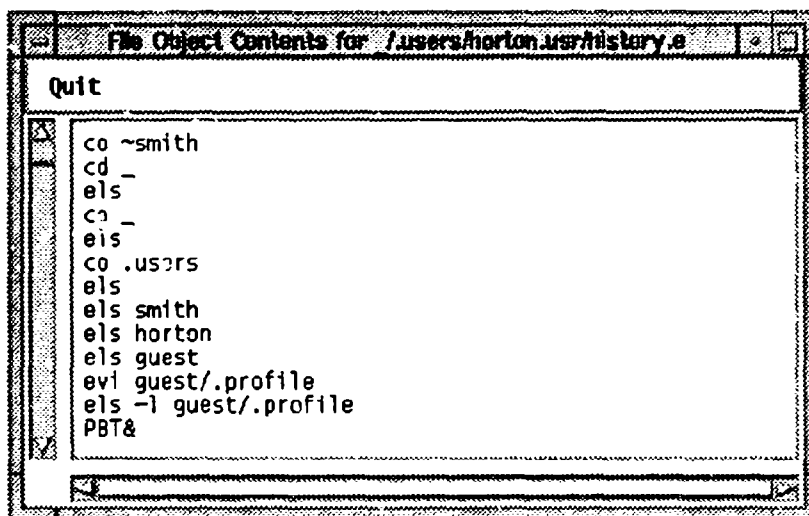


Figure 45: PCTE File Contents Text Window

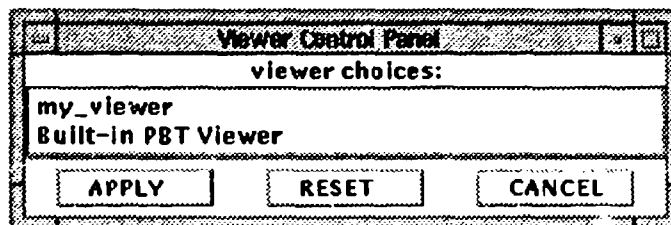


Figure 46: Viewer Selection Control Panel

object's pathname.

4.8.4 Invoking Actions on PCTE Objects

Section 4.8.3 described how the existence of `pbt-viewer` links emanating from file objects can be used to invoke display programs that are (presumably) specialized for the type of file object. The PBT also supports a more generalized mechanism that can be used for *any* type of PCTE object—via `pbt-action` links. Like `pbt-viewer` links, `pbt-action` links terminate at *static contexts*. However, the source of a `pbt-action` can be any PCTE object.

These *action* programs are invoked by selecting the **Perform Action** menu item in any object's popup menu. However, if an object has no such action links emanating from it, then this menu item is “desensitized” and, therefore, cannot be selected. Figure 47 shows what the menu will look like if at least one action exists. When the **Perform Action** is selected, the PBT pops up an Action Selection Control Panel, such as the one shown in figure 48. This allows the user to choose which action program to invoke.

The mechanism for invoking the action tool is the same as for invoking viewer tools when

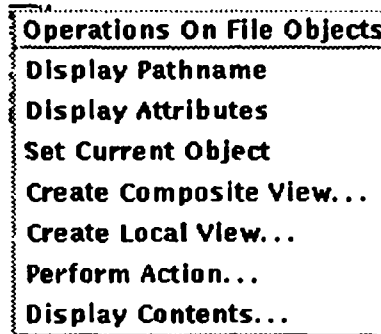


Figure 47: Object Menu With Actions

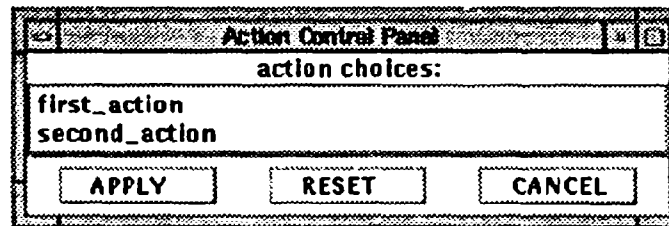


Figure 48: Action Selection Control Panel

displaying file contents:

If the “tool” is an ASCII text file, then it is assumed to be an `esh` script, and the shell will be invoked to interpret the script. Otherwise, the tool will be invoked directly.

The pathname of the `pbt-action` link’s source object is passed to the invoked program.

If the `pbt-action` link has a non-null value for the (string) `pre_args` attribute, then this attribute value will be passed to the invoked program as the argument preceding the source object’s pathname.

If the `pbt-viewer` link has a non-null value for the (string) `post_args` attribute, then this attribute value will be passed to the invoked program as the argument following the source object’s pathname.

4.9 Link-Oriented Operations

The PBT currently supports the following operations on links found within View windows:

- Displaying the pathname within that View of the link’s target object.
- Displaying the link’s non-key attributes (if any).

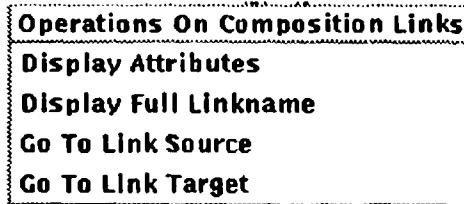


Figure 49: Link Operations Menu

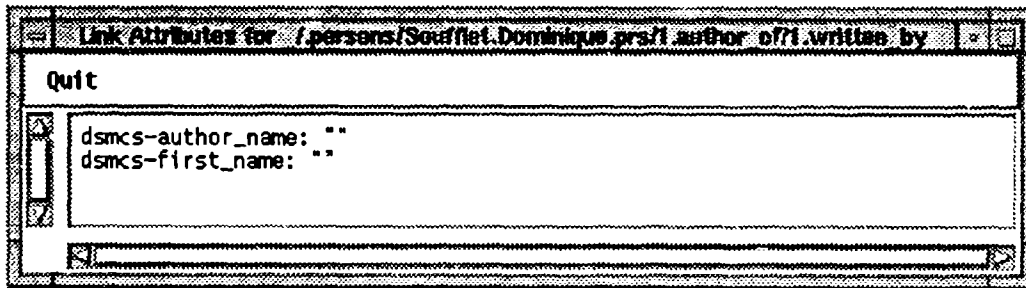


Figure 50: Link Attribute Values Text Window

- Scrolling the graph such that the link's source object is approximately centered within the View window.
- Scrolling the graph such that the link's target object is approximately centered within the View window.

All of these operations are accessed by selecting the icon associated with the link. This pops up an **Operations on Links** Menu, such as the one for composition links shown in figure 49.

4.9.1 Displaying Non-Key Link Attributes

If the **Display Attributes** menu item is selected, the PBT will pop up a text window such as is shown in figure 50; however, if there are no non-key attributes for the link, then an alert box will be popped up instead.

4.9.2 Displaying Full Linkname

If the **Display Full Linkname** menu item is selected, a text window such as is shown in figure 51 will pop up, containing the pathname by which the link's target object is known in that View.

- The maximum length to display sections of pathnames before breaking the name onto another line.
- Whether to display PCTE exception information automatically when PCTE exceptions are raised during the processing of PBT commands.
- Whether “delete View” and/or “terminate the browser session” requests need to be confirmed.

As discussed in section 4.3.2, the user can control the maximum distance from the start object whenever a Local View is created by modifying the “maximum distance” text field of the Local View Control Panel (see figure 20). The PBT gives the user control over the initial value of the “maximum distance” field. 2 is the “built-in” default maximum distance.

As discussed in section 4.3.1, the user can control the maximum depth of a composition tree whenever a Composite View is created by modifying the “maximum depth” text field of the Composite View Control Panel (see figure 19). The PBT gives the user control over the initial value of the “maximum depth” field. (NOTE: 2 is the “built-in” default maximum depth.)

Another aspect of Composite View creation is the specification of the kinds of links to be included in the Views. This is controlled via the “link kinds” radio buttons in the Composite View Control Panel. The PBT gives the user control over which radio button is initially set. The “built-in” link kinds value is *composition links only*.

As seen in all of the Local and Composite View window examples above, SDS names are not used within node and arc labels. However, the PBT allows the user to modify this default PBT behavior. Figure 52 contains a View created when SDS name inclusion has been requested.

When object pathnames are displayed in text windows such as that of figure 43, the PBT breaks each pathname when sections of the name exceed a “maximum pathname segment length” number of characters. By default, this cutoff length value is 72.

Normally (i.e., by default) when PCTE exceptions are raised during the execution of a user-initiated PBT action, an alert box is raised to inform the user of a problem, but the exact PCTE exception is ignored. (See section 4.10 on displaying the most recent PCTE exception information.) The user may also indicate that the type of PCTE exception be automatically displayed (in the *xterm* window from which the PBT was invoked) when these exceptions are detected.

When object pathnames are displayed in text windows such as that of figure 43, the PBT breaks each pathname when sections of the name exceed 72 characters. However, the PBT allows the user to modify this default length value.

Normally (i.e., by default) when PCTE exceptions are raised during the execution of a user-initiated PBT action, an alert box is raised to inform the user of a problem, but the exact

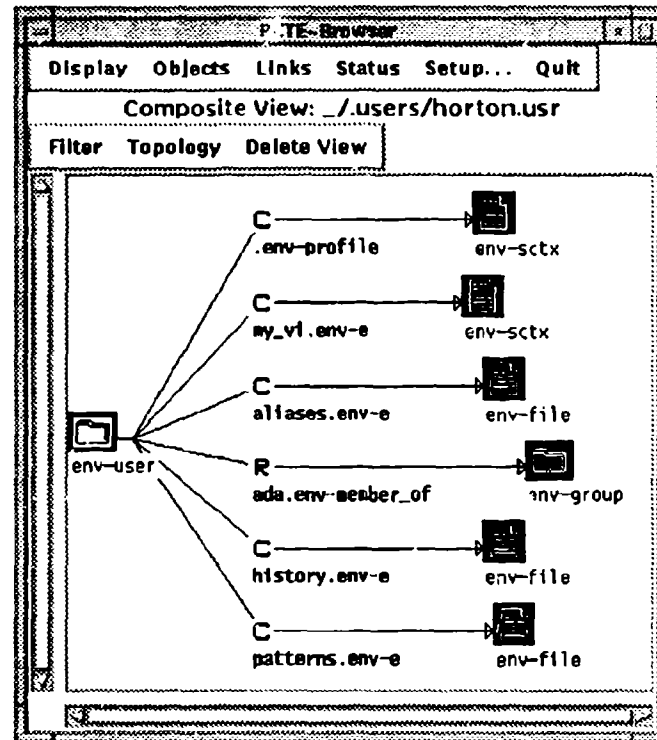


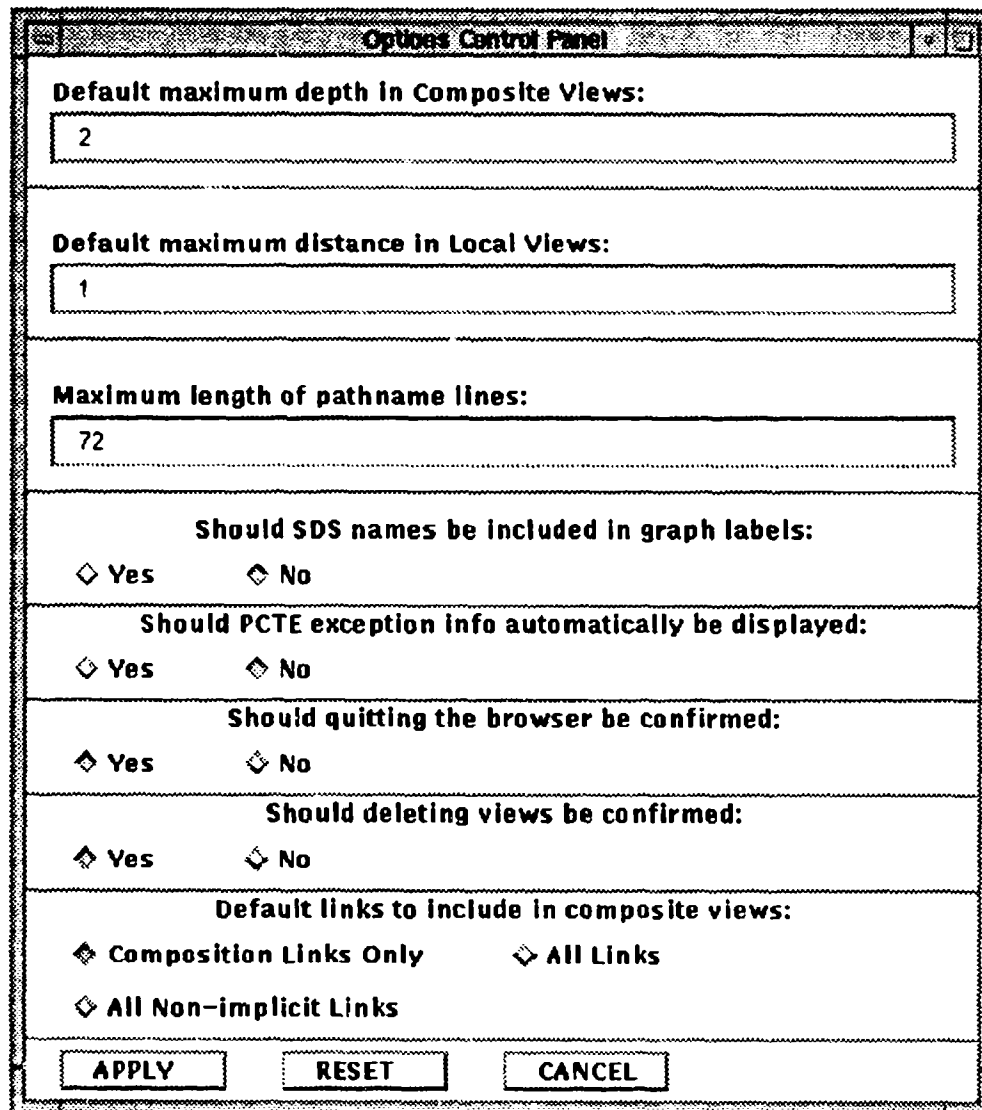
Figure 52: Graph Including SDS Names

PCTE exception is ignored. (See section 4.10 on how to display the most recent PCTE exception information.) However, the user may instead indicate that the type of PCTE exception always be displayed (in the **xterm** window from which the PBT was invoked) when these exceptions are detected.

By default, all requests to delete specific View windows and requests to terminate the entire PBT session must be confirmed by the user, to prevent unintentional operations. This is done for the benefit of less experienced PBT users. However, for users with more experience (or less patience), the PBT allows the user to "turn off" the confirmation processing.

All of the customizable aspects of PBT behavior can be changed dynamically, during a PBT session, via the **Setup** command: Clicking on the global **Setup** command button in pops up the Setup Control Panel shown in figure 53. Changing specific components within this panel and applying the changes will effect the rest of the PBT session. Note that these changes will affect *all* View windows, not just the one whose global menu bar was used to pop up the control panel.

Any or all of the customizable aspects of PBT behavior can also be set via PBT command line arguments, as follows:



The image shows a Windows-style dialog box titled "Options Control Panel". It contains several sections for configuring the application's behavior. Each section has a label followed by a text input field or a set of radio buttons. The sections are: "Default maximum depth in Composite Views:" with a value of 2; "Default maximum distance in Local Views:" with a value of 1; "Maximum length of pathname lines:" with a value of 72; "Should SDS names be included in graph labels:" with radio buttons for Yes and No; "Should PCTE exception info automatically be displayed:" with radio buttons for Yes and No; "Should quitting the browser be confirmed:" with radio buttons for Yes and No; "Should deleting views be confirmed:" with radio buttons for Yes and No; and "Default links to include in composite views:" with radio buttons for Composition Links Only, All Links, and All Non-implicit Links. At the bottom of the dialog are three buttons: APPLY, RESET, and CANCEL.

Options Control Panel	
Default maximum depth in Composite Views:	
2	
Default maximum distance in Local Views:	
1	
Maximum length of pathname lines:	
72	
Should SDS names be included in graph labels:	
<input type="radio"/> Yes	<input type="radio"/> No
Should PCTE exception info automatically be displayed:	
<input type="radio"/> Yes	<input type="radio"/> No
Should quitting the browser be confirmed:	
<input type="radio"/> Yes	<input type="radio"/> No
Should deleting views be confirmed:	
<input type="radio"/> Yes	<input type="radio"/> No
Default links to include in composite views:	
<input type="radio"/> Composition Links Only	<input type="radio"/> All Links
<input type="radio"/> All Non-implicit Links	
APPLY RESET CANCEL	

Figure 53: Setup Control Panel

The `-i<distance>` argument will set the default maximum distance for Local Views. For example, the following argument will set this maximum distance to 3:

`-i3`

The `-k<link types>` argument will set the default types of links to include in Composite Views. The `link_types` component of the `-k` argument can take on the same set of values as it can for the `-o` argument (see section 4.3.1.2), namely: the following values:

- `c` (i.e., `-kc`) will include composition links only
- `n` (i.e., `-kn`) will include all non-implicit links
- `a` (i.e., `-ka`) will include all types of links

The `-e<depth>` argument will set the default maximum depth for Composite Views. For example, the following argument will set this maximum distance to -1 (meaning *no* maximum depth is used):

`-e-1`

The `-p<length>` argument will set the maximum length of pathname line sections when using the “display object pathname” commands. For example, the following argument will set this maximum length to 50:

`-p50`

All the other customizable aspects of PBT behavior fall into the *yes or no* category, i.e., *yes* the PBT does something, or *no* it does not. To set or reset these aspects, the `-y<flags>` and `-n<flags>` arguments can be used (where `-y` indicates that “yes, the corresponding flags should be considered set,” and `-n` indicates that “no, they should be considered reset”). The allowable `<flags>` values are the following:

- `s` — controls the automatic display of PCTE exception information
- `q` — controls the confirming of “terminate the browser” requests
- `d` — controls the confirming of “delete View” requests
- `i` — controls the inclusion of SDS names in View icon labels, etc.

For example, the consider the following command line arguments:

`-nqd -yis`

These two arguments modify all of of these *yes or no* types of customizable behaviors: “No,” don’t confirm either terminate or delete requests, but “yes,” include SDS names in Views, and “yes,” automatically display exception information.

4.12 Terminating the Browser Session

Clicking the **Quit** button any View window destroys all browser windows created in that PBT session, and terminates the browser.

A Appendix: Customizing the PBT X Resources

The following is the current contents of the color-oriented X Resource File PCTE-Browser.color. (In the version of this file for a black and white display, PCTE-Browser.black_n.white, all of the lines specifying *foreground* and *background* colors have been removed.)

```
#--|
#--|Item: PCTE Browser Tool (PBT)
#--|$Revision:"1.0"$
#--|$State: Prototype $
#--|
#--|The Version Description Document (VDD) included with this release provides
#--|detailed information regarding the condition of the software. The "User
#--|Feedback" section of the VDD describes how to obtain additional information.
#--|
#--|Distribution Statement "A", per DoD Directive 5230.24
#--|Authorized for public release; distribution is unlimited.
#--|
#--|Copyright (c) 1992, Paramax Systems Corporation, Reston, Virginia
#--|Copyright is assigned to the U.S. Government, upon delivery thereto,
#--|in accordance with the DFAR Special Works Clause.
#--|
#--|Developed by: Paramax Systems Corporation,
#--|
#--|This software, developed under the Software Technology for Adaptable,
#--|Reliable Systems (STARS) program, is approved for release under
#--|Distribution "A" of the Scientific and Technical Information Program
#--|Classification Scheme (DoD Directive 5230.24) unless otherwise indicated.
#--|Sponsored by the U.S. Defense Advanced Research Projects Agency (DARPA)
#--|under contract F19628-88-D-0031, the STARS program is supported by the
#--|military services, SEI, and MITRE, with the U.S. Air Force as the executive
#--|contracting agent.
#--|
#--|Permission to use, copy, modify, and comment on ' ' software and its
#--|documentation for purposes stated under Distribution "A" and without fee
#--|is hereby granted, provided that this notice appears in each whole or
#--|partial copy. This software retains Contractor indemnification to
#--|the Government regarding copyrights pursuant to the above referenced
#--|STARS contract. The Government disclaims all responsibility against
#--|liability, including costs and expenses for violation of proprietary
#--|rights, or copyrights arising out of the creation or use of this
#--|software.
#--|
#--|In addition, the Government, Paramax, and its subcontractors disclaim all
```

#--|warranties with regard to this software, including all implied warranties
#--|of merchantability and fitness, and in no event shall the Government,
#--|Paramax, or its subcontractor(s) be liable for any special, indirect or
#--|consequential damages or any damages whatsoever resulting from the loss of
#--|use, data, or profits, whether in action of contract, negligence or other
#--|tortious action, arising in connection with the use or performance of this
#--|software.

#--|

#--|\$Log \$

#--|

#-- Version Information

#-- ~~~~~

#--

#-- PCTE Browser Tool (PBT).

#-- Version: 1.0.

#-- Release Date: November 30, 1992.

#-- Compiled under SUN OS 4.1.2 using SunAda 1.0

#--

#--

#-- General Information

#-- ~~~~~

#--

#-- This is a Motif-oriented release of the Paramax STARS PCTE Browser Tool,
#-- which can graphically display user-specified sections of a PCTE object
#-- base. It is an instance of the Paramax STARS Reusable Graphical Browser
#-- (RGB), which itself makes use of the (commercial) SA-Motif Ada bindings
#-- from Systems Engineering Research Corporation (SERC). The PBT is currently
#-- operational on Sun-4 workstations under the X Window System, running the
#-- Emeraude PCTE implementation, release 12.2 or 12.3.
#-- See the PBT release 1.0 Users Manual for information on its capabilities.

#--

#-- For further information, contact the authors:

#--

#-- Michael Horton
#-- Paramax Systems Corp.
#-- Valley Forge Labs
#-- 70 E. Swedesford Road
#-- Paoli, PA. 19301
#-- horton@prc.unisys.com (current)
#-- horton@vfl.paramax.com (future)
#-- +1 215 648-2527

#--

#-- Robert Smith
#-- Paramax Systems Corp.
#-- Valley Forge Labs

```

#--          70 E. Swedesford Road
#--          Paoli, PA. 19301
#--          smith@prc.unisys.com (current)
#--          smith@vfl.paramax.com (future)
#--          +1 215 648-2402
#####
# Resource file for the PCTE Browser Tool application
#####
# RGB resource file name = PCTE-Browser
# RGB resource directory path = $XAPPLRESDIR

# customize the views
#-----
vshell*scr_window.width: 500
vshell*scr_window.height: 400
vshell*scr_window.scrollingPolicy: AUTOMATIC
vshell*scr_window.scrollBarPlacement: BOTTOM_LEFT
vshell*scr_window.scrollBarDisplayPolicy: STATIC
vshell*scr_window.bottomAttachment: attach_form

vshell*fontList: lucidasans-bold-12
vshell*title*fontList: lucidasans-bold-14

vshell*menubar_1*adjustLast: TRUE
vshell*menubar_2*adjustLast: TRUE

vshell*tp_bottom.scrollingPolicy: AUTOMATIC
vshell*tp_bottom.scrollBarPlacement: BOTTOM_LEFT
vshell*tp_bottom.scrollBarDisplayPolicy: STATIC

vshell*tp_right.scrollingPolicy: AUTOMATIC
vshell*tp_right.scrollBarPlacement: BOTTOM_RIGHT
vshell*tp_right.scrollBarDisplayPolicy: STATIC

# customize the nodes
#-----
vshell*scr_window*node_FILE.background: red
vshell*scr_window*node_FILE.foreground: white
vshell*scr_window*node_FILE.labelType: PIXMAP
vshell*scr_window*node_FILE.labelPixmap: f_node.xbm
vshell*scr_window*node_FILE.reverseLabelPixmap: f_node_inv.xbm
vshell*scr_window*node_FILE.borderWidth: 0
vshell*scr_window*node_FILE.internalWidth: 0
vshell*scr_window*node_FILE.internalHeight: 0
vshell*scr_window*node_FILE.bottomShadowColor: black

```

```
vshell*scr_window*node_FILE.topShadowColor: gray
vshell*scr_window*node_label_FILE.borderWidth: 0
vshell*scr_window*node_label_FILE.fontList: lucidasanstypewriter-12
vshell*scr_window*node_attr_FILE.height: 50
vshell*scr_window*node_attr_FILE.width: 100

vshell*scr_window*node_PROCESS.labelType: PIXMAP
vshell*scr_window*node_PROCESS.labelPixmap: p_node.xbm
vshell*scr_window*node_PROCESS.reverseLabelPixmap: p_node_inv.xbm
vshell*scr_window*node_PROCESS.borderWidth: 0
vshell*scr_window*node_PROCESS.internalWidth: 0
vshell*scr_window*node_PROCESS.internalHeight: 0
vshell*scr_window*node_PROCESS.bottomShadowColor: black
vshell*scr_window*node_PROCESS.topShadowColor: gray
vshell*scr_window*node_label_PROCESS.borderWidth: 0
vshell*scr_window*node_label_PROCESS.fontList: lucidasanstypewriter-12
vshell*scr_window*node_attr_PROCESS.height: 50
vshell*scr_window*node_attr_PROCESS.width: 100

vshell*scr_window*node_OBJECT.background: goldenrod
vshell*scr_window*node_OBJECT.foreground: white
vshell*scr_window*node_OBJECT.labelType: PIXMAP
vshell*scr_window*node_OBJECT.labelPixmap: o_node.xbm
vshell*scr_window*node_OBJECT.reverseLabelPixmap: o_node_inv.xbm
vshell*scr_window*node_OBJECT.borderWidth: 1
vshell*scr_window*node_OBJECT.internalWidth: 0
vshell*scr_window*node_OBJECT.internalHeight: 0
vshell*scr_window*node_OBJECT.bottomShadowColor: black
vshell*scr_window*node_OBJECT.topShadowColor: gray
vshell*scr_window*node_label_OBJECT.borderWidth: 0
vshell*scr_window*node_label_OBJECT.fontList: lucidasanstypewriter-12
vshell*scr_window*node_attr_OBJECT.height: 50
vshell*scr_window*node_attr_OBJECT.width: 100

vshell*scr_window*node_GROUP.labelType: PIXMAP
vshell*scr_window*node_GROUP.labelPixmap: g_node.xbm
vshell*scr_window*node_GROUP.reverseLabelPixmap: g_node_inv.xbm
vshell*scr_window*node_GROUP.borderWidth: 0
vshell*scr_window*node_GROUP.internalWidth: 0
vshell*scr_window*node_GROUP.internalHeight: 0
vshell*scr_window*node_GROUP.bottomShadowColor: black
vshell*scr_window*node_GROUP.topShadowColor: gray
vshell*scr_window*node_label_GROUP.borderWidth: 0
vshell*scr_window*node_label_GROUP.fontList: lucidasanstypewriter-12
vshell*scr_window*node_attr_GROUP.height: 50
```

```
vshell*scr_window*node_attrib_GROUP.width: 100

vshell*scr_window*node_SDS.labelType: PIXMAP
vshell*scr_window*node_SDS.labelPixmap: s_node.xbm
vshell*scr_window*node_SDS.reverseLabelPixmap: s_node_inv.xbm
vshell*scr_window*node_SDS.borderWidth: 0
vshell*scr_window*node_SDS.internalWidth: 0
vshell*scr_window*node_SDS.internalHeight: 0
vshell*scr_window*node_SDS.bottomShadowColor: black
vshell*scr_window*node_SDS.topShadowColor: gray
vshell*scr_window*node_label_SDS.borderWidth: 0
vshell*scr_window*node_label_SDS.fontList: lucidasanstypewriter-12
vshell*scr_window*node_attrib_SDS.height: 50
vshell*scr_window*node_attrib_SDS.width: 100

# customize the arcs
#-----
vshell*scr_window*arc_COMPOSITION_LINK.labelType: pixmap
vshell*scr_window*arc_COMPOSITION_LINK.labelPixmap: c_rel.xbm
vshell*scr_window*arc_COMPOSITION_LINK.reverseLabelPixmap: c_rel_inv.xbm
vshell*scr_window*arc_label_COMPOSITION_LINK.border_width: 0
vshell*scr_window*arc_label_COMPOSITION_LINK.fontList: lucidasanstypewriter-bold-12
vshell*scr_window*arc_attrib_COMPOSITION_LINK.height: 30
vshell*scr_window*arc_attrib_COMPOSITION_LINK.width: 150

vshell*scr_window*arc_REFERENCE_LINK.labelType: pixmap
vshell*scr_window*arc_REFERENCE_LINK.labelPixmap: r_rel.xbm
vshell*scr_window*arc_REFERENCE_LINK.reverseLabelPixmap: r_rel_inv.xbm
vshell*scr_window*arc_label_REFERENCE_LINK.border_width: 0
vshell*scr_window*arc_label_REFERENCE_LINK.fontList: lucidasanstypewriter-bold-12
vshell*scr_window*arc_attrib_REFERENCE_LINK.height: 30
vshell*scr_window*arc_attrib_REFERENCE_LINK.width: 150

vshell*scr_window*arc_IMPLICIT_LINK.labelType: pixmap
vshell*scr_window*arc_IMPLICIT_LINK.labelPixmap: i_rel.xbm
vshell*scr_window*arc_IMPLICIT_LINK.reverseLabelPixmap: i_rel_inv.xbm
vshell*scr_window*arc_label_IMPLICIT_LINK.border_width: 0
vshell*scr_window*arc_label_IMPLICIT_LINK.fontList: lucidasanstypewriter-bold-12
vshell*scr_window*arc_attrib_IMPLICIT_LINK.height: 30
vshell*scr_window*arc_attrib_IMPLICIT_LINK.width: 150

vshell*scr_window*arc_EXISTENCE_LINK.labelType: pixmap
vshell*scr_window*arc_EXISTENCE_LINK.labelPixmap: e_rel.xbm
vshell*scr_window*arc_EXISTENCE_LINK.reverseLabelPixmap: e_rel_inv.xbm
vshell*scr_window*arc_label_EXISTENCE_LINK.border_width: 0
```

```
vshell*scr_window*arc_label_EXISTENCE_LINK.fontList: lucidasanstypewriter-bold-12
vshell*scr_window*arc_attrib_EXISTENCE_LINK.height: 30
vshell*scr_window*arc_attrib_EXISTENCE_LINK.width: 150

vshell*scr_window*arc_DESIGNATION_LINK.labelType: pixmap
vshell*scr_window*arc_DESIGNATION_LINK.labelPixmap: d_rel.xbm
vshell*scr_window*arc_DESIGNATION_LINK.reverseLabelPixmap: d_rel_inv.xbm
vshell*scr_window*arc_label_DESIGNATION_LINK.border_width: 0
vshell*scr_window*arc_label_DESIGNATION_LINK.fontList: lucidasanstypewriter-bold-12
vshell*scr_window*arc_attrib_DESIGNATION_LINK.height: 30
vshell*scr_window*arc_attrib_DESIGNATION_LINK.width: 150

vshell*scr_window*arc_VIEWER_REFERENCE_LINK.labelType: pixmap
vshell*scr_window*arc_VIEWER_REFERENCE_LINK.labelPixmap: v_rel.xbm
vshell*scr_window*arc_VIEWER_REFERENCE_LINK.reverseLabelPixmap: v_rel_inv.xbm
vshell*scr_window*arc_label_VIEWER_REFERENCE_LINK.border_width: 0

vshell*scr_window*arc_label_VIEWER_REFERENCE_LINK.fontList: lucidasanstypewriter-bold-12
vshell*scr_window*arc_attrib_VIEWER_REFERENCE_LINK.height: 30
vshell*scr_window*arc_attrib_VIEWER_REFERENCE_LINK.width: 150

vshell*scr_window*arc_ACTION_REFERENCE_LINK.labelType: pixmap
vshell*scr_window*arc_ACTION_REFERENCE_LINK.labelPixmap: a_rel.xbm
vshell*scr_window*arc_ACTION_REFERENCE_LINK.reverseLabelPixmap: a_rel_inv.xbm
vshell*scr_window*arc_label_ACTION_REFERENCE_LINK.border_width: 0
vshell*scr_window*arc_attrib_ACTION_REFERENCE_LINK.height: 30
vshell*scr_window*arc_attrib_ACTION_REFERENCE_LINK.width: 150

vshell*scr_window*arc_label_ACTION_REFERENCE_LINK.fontList: lucidasanstypewriter-bold-12

# customize outdated flag icons
#-----
vshell*outdated_flag_up.bitmap: outdated.xbm
vshell*node_label_SDS.borderWidth: 0

# customize popup text windows
#-----
*tshell.width: 550
*tshell.height: 300
*tshell*fontList: lucidasanstypewriter-12
*tshell*text_menubar*fontList: lucidasanstypewriter-bold-12

# customize alert windows
#-----
*ashell*fontList: lucidasans-bold-12
```

```
*ashell.width: 300
*ashell.height: 50

# customize popup dialog windows
#-----
*psell*keyboardFocusPolicy: POINTER

# customize control panel windows
#-----
*cshell*fontList: lucidasans-bold-12
*cshell*sb_text.fontList: lucidasanstypewriter-12

*cshell*scrollBarPlacement: BOTTOM_LEFT

*cshell*keyboardFocusPolicy: POINTER

*cshell*buttons*adjustLast: FALSE
*cshell*buttons*adjustMargin: TRUE
*cshell*buttons*packing: PACK_COLUMN
*cshell*buttons*spacing: 25
*cshell*buttons*marginWidth: 15
*cshell*radio_button_field*marginWidth: 20

# customize "dialog with history" windows
#-----
*dshell.width: 450

*dshell*buttons*bottomAttachment: ATTACH_FORM
*dshell*buttons*packing: PACK_COLUMN
*dshell*buttons*adjustLast: FALSE
*dshell*buttons*marginWidth: 15

*dshell*historySW.rightAttachment: ATTACH_FORM
*dshell*historySW.leftAttachment: ATTACH_FORM
*dshell*historySW.scrollBarPlacement: BOTTOM_LEFT

*dshell*scrollBarPlacement: BOTTOM_LEFT
*dshell*scrollBarDisplayPolicy: AS_NEEDED
*dshell*scrollingPolicy: AUTOMATIC

*dshell*keyboardFocusPolicy: POINTER

# set colors of browser components
#-----
*background: turquoise
```

*sb_text.background: white

vshell*menubar_1*background: pink

vshell*title*background: white

vshell*menubar_2*background: pink

vshell*ScrolledWindowClipWindow.background: white

vshell*bb.background: white

vshell*scr_window*node_label_FILE.background: white

vshell*scr_window*node_label_FILE.foreground: black

vshell*scr_window*node_label_PROCESS.background: white

vshell*scr_window*node_label_PROCESS.foreground: black

vshell*scr_window*node_label_OBJECT.background: white

vshell*scr_window*node_label_OBJECT.foreground: black

vshell*scr_window*node_label_GROUP.background: white

vshell*scr_window*node_label_GROUP.foreground: black

vshell*scr_window*node_label_SDS.background: white

vshell*scr_window*node_label_SDS.foreground: black

vshell*scr_window*arc_label_COMPOSITION_LINK.background: white

vshell*scr_window*arc_label_COMPOSITION_LINK.foreground: black

vshell*scr_window*arc_label_REFERENCE_LINK.background: white

vshell*scr_window*arc_label_REFERENCE_LINK.foreground: black

vshell*scr_window*arc_label_IMPLICIT_LINK.background: white

vshell*scr_window*arc_label_IMPLICIT_LINK.foreground: black

vshell*scr_window*arc_label_EXISTENCE_LINK.background: white

vshell*scr_window*arc_label_EXISTENCE_LINK.foreground: black

vshell*scr_window*arc_label_DESIGNATION_LINK.background: white

vshell*scr_window*arc_label_DESIGNATION_LINK.foreground: black

vshell*scr_window*arc_label_VIEWER_REFERENCE_LINK.background: white

vshell*scr_window*arc_label_VIEWER_REFERENCE_LINK.foreground: black

vshell*scr_window*arc_label_ACTION_REFERENCE_LINK.background: white

vshell*scr_window*arc_label_ACTION_REFERENCE_LINK.foreground: black

*cshell*single_selection_list*background: white

*cshell*multi_selection_list*background: white

*tshell*scrolled_text.background: white

*defaultVirtualBindings: \

osfBackSpace : <Key>BackSpace\n\

osfInsert : <Key>Insert\n\

osfDelete : <Key>Delete\n\

BMenu : <Btn1>

B Appendix: PCTE Browser Tool Help Message

The following is the output of the PBT when invoked using the -h command line argument:

Available command line arguments:

- h prints this help message, and terminates the PBT session.
- l[<distance>] [<pathname>] creates an initial Local View starting at object <pathname>. If the maximum <distance> was omitted, then the default distance will be used. If the <pathname> was omitted, then the initial current object will be used.
- o[<link_types>][<depth>] [<pathname>] creates an initial Composite View starting at object <pathname>. If the maximum <depth> of composite tree was omitted, then the default maximum depth will be used. The <link types> value, if included, indicates which types of links to include in the graph:
 - c (i.e., -oc) -- composition links only;
 - n (i.e., -on) -- all non-implicit links;
 - a (i.e., -oa) -- all types of links.if the <link types> is omitted, then the default link types restriction will be used. If the <pathname> was omitted, then the initial current object will be used.
- s <working_schema> specifies the list of SDSes to be used as the initial working schema.
- c <pathname> specifies the pathname of the object to be used as the initial Current_Object.
- e<depth> specifies the default maximum depth value to be used when creating Composite Views.
- i<distance> specifies the default maximum distance value to be used when creating Local Views[214z.
- p<length> specifies the default maximum length of lines to be used when displaying pathnames.
- y[<flag_list>] means that all of the PBT options specified in the flag list should be set on. Valid flags include:
 - s -- automatically display PCTE error info
 - q -- requests to quit the browser should be confirmed
 - d -- requests to delete views should be confirmed
 - i -- SDS names should be included when displaying link type names and object type namesIf the flag list is omitted, then ALL of the PBT options will be set on.

- n[<flag_list>] means that all of the PBT options specified in the flag list should be set off. The allowable flags are the same as for the "-y" argument.
If the flag list is omitted, then ALL of the PBT options will be set of.
- k<link_types> specifies the types of links to include by default when creating Composite Views. The allowable link types are the same as in the "-o" argument:
- c (i.e., -kc) -- composition links only;
 - n (i.e., -kn) -- all non-implicit links;
 - a (i.e., -ka) -- all types of links.
- a <pathname> specifies the pathname of the object from which to read the initial set of object aliases.
- t <pathname> specifies the pathname of the object from which to read the initial set of link name patterns.